VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Mobilní aplikace pro podporu specifické profese

Mobile Applications to Support Specific Profession

2015 Jakub Urbiš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jakub Urbiš**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Mobilní aplikace pro podporu specifické profese
Mobile Applications to Support Specific Profession

Zásady pro vypracování:

Cílem bakalářské práce je vytvoření mobilní aplikace pro podporu procesů specifické profese, a to taxikář. Student vyhodnotí a popíše potřeby této profese se specifickými nároky na plánování a řízení její činnosti. Výsledná aplikace bude oporou pro činnost taxikáře a umožní komplexní organizaci a plánování činností této profese.

1. Popis vývojového prostředí Android Studio BETA použitého při vývoji a popis práce s tímto prostředím.
2. Úvod do procesního řízení (Business Process Management - BPM).
3. Analýza a návrh aplikace pro mobilní systém Android 4.
4. Tvorba mobilní aplikace na základě návrhu.
5. Vytvořit uživatelskou příručku k aplikaci.

Aplikace bude obsahovat:
1. Bázi znalostí a procesů nutných k základní funkci (počítání ujetých kilometrů, výpočet nákladů na jízdu, atd...).
2. Editor znalostí a procesů pro editaci proměnných (spotřeba, cena, atd...), a změnu či tvorbu nových uživatelských procesů.

Seznam doporučené odborné literatury:

[1] Florian John Sowa, Knowledge representation: Logical, Philosophical, and Computational Foundations. Brooks Cole Publishing Co., USA, 2000, ISBN: 0-534-94965-7
[2] Wei-Meng Lee. Beginning Android 4 Application Development, Wrox Press, 2012, ISBN: 978-1-118-19954-1

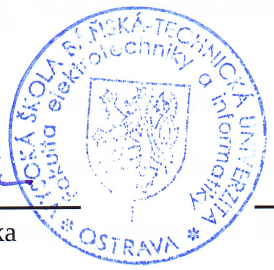
Další literatura podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Břetislav Paláček**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2015 .....

Velmi rád bych touto cestou poděkoval panu Ing. Břetislavu Paláčkovi za podnětnou diskuzi, cenné rady a ochotu v průběhu tvorby práce. Také bych chtěl poděkovat svým rodičům za všechnu podporu, které se mi během studia dostalo.

Abstrakt

Cílem bakalářské práce je vytvoření mobilní aplikace pro podporu profese taxikáře. Čtenář je dále seznámen se základy procesního řízení - BPM a základními prvky při modelování procesů a návrhu softwaru, které se následně využijí při návrhu aplikace. Bakalářská práce prezentuje základní aspekty vývoje pro platformu Android ve vývojovém prostředí Android Studio. Aplikace bude vyvíjena na základě vytvořeného návrhu a je součástí přílohy, jakožto i uživatelská příručka k této aplikaci.

Klíčová slova: Android, Android Studio, Java, BPM, analýza, návrh, vývoj, mobilní aplikace

Abstract

The aim of this bachelor thesis is development of a mobile application to support the profession of a taxi driver. The reader is also familiarised with the basics of process management - BPM and basic components used during making business models and designing software, which will be used afterwards. The bachelor thesis presents the fundamental aspects of a development for the Android platform in development environment Android Studio. The application will be based on the created design. Application and user guide are included in Annex.

Keywords: Android, Android Studio, Java, BPM, analysis, design, development, mobile applications

Seznam použitých zkratek a symbolů

BMP	–	Business Process Management
UML	–	Unified Modeling Language
CAD	–	Computer-Aided Design
IDE	–	Integrated Development Environment
ADT	–	Android Developer Tools
JDK	–	Java Development Kit
SDK	–	Software Development Kit
XML	–	Extensible Markup Language
OS	–	Operating System
AVD	–	Android Virtual Device
USB	–	Universal Serial Bus
API	–	Application Programming Interface
SD	–	Secure Digital
GPS	–	Global Positioning System
URI	–	Uniform Resource Identifier
JSON	–	JavaScript Object Notation

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Bakalářská práce má za cíl zjednodušit každodenní činnosti specifické profese, a to profese taxikáře formou mobilní aplikace. Jelikož lze vyhodnotit a popsat potřeby s různými nároky na plánování a řízení jejich činnosti, lze také vytvořit aplikaci, která by byla oporou právě pro tyto činnosti. Rozhodl jsem se aplikaci vyvíjet pro systém Android, který je dnes nerozšířenějším systémem pro mobilní zařízení. Jako vývojové prostředí jsem si zvolil Android Studio (v době vývoje ještě BETA), které je doporučováno přímo společností Google a které usnadňuje samotnou práci.

První kapitolou práce je letmý úvod do procesního řízení - BPM a jeho základní vlastnosti. Nalezneme zde popis a charakteristiku procesů a také, jak se jednotlivé procesy třídí podle nejrůznějších kritérií. Podkapitola věnující se UML a části popisující diagramy spadající do této oblasti (třídní, sekvenční a jiné) je důležitá pro kapitolu návrhu aplikace.

Po úvodu do procesního řízení následuje kapitola popisující vývojové prostřední Android Studio, které bylo použito při vývoji. Nejprve čtenáře seznamuji s možnostmi pro vývoj Android aplikací a shrnuji klady a zápory Android Studia. Dále kapitola obsahuje základní prvky vývoje v Android Studiu, výhody a "features", které přináší právě toto IDE a prvky, které jsou charakteristické pro Android aplikace.

Následující kapitola je věnována analýze a návrhu aplikace pro mobilní systém Android 4. V kapitole nejprve shrnuji zadání a specifikace požadavků na vytvářenou aplikaci. Dozvíme se také, proč jsem zvolil systém Android. Další odstavce jsou věnovány detailnímu návrhu případu užití a také třídnímu diagramu, který zobrazuje jádro aplikace. Třídní diagram je dále doplněn informacemi o jednotlivých třídách a jejich významu v rámci aplikace. Následuje sekvenční diagram zachycující vytváření nového procesu a vývojový diagram ukazující, jak je modelovaný proces zobrazen uživateli.

O tvorbě mobilní aplikace na základě návrhu pojednává poslední kapitola. Zde postupně ukazuji implementaci funkcí, které jsou definované při návrhu a analýze požadavků. Vysvětluji zde postup práce synchronizace s kalendářem od Google, ukládání výsledků namodelovaných procesů do externího úložiště ve formátu JSON. Objasňuji také způsob zápisu v tomto formátu. Následuje podkapitola věnující se práce s notifikacemi a souvisejícími třídami a podkapitoly popisující modelování nového procesu a výpočty aplikace.

Součástí práce je vytvořená mobilní aplikace pro systém Android 4 a také příručka pro práci s touto aplikací.

2 Úvod do procesního řízení

2.1 Procesní řízení

Pokud potřebujeme optimalizovat procesy, ať už jde o vyřízení obchodních faktur, žádosti o dovolenou, vyřízení pojistné události nebo jen nákup potřebných surovin, využijeme procesního řízení. Procesní řízení (také BPM – Business Process Management) nám poskytuje prostředky pro plánování, řízení, optimalizaci, automatizaci a sledování firemních i zákaznických procesů. Pokud pracují zaměstnanci (osoby), firemní aplikace či informační systémy a sdílený obsah ve vzájemném souladu, jsou splněny nutné podmínky produktivity a efektivity procesů.

Procesní řízení tedy využívá znalostí, dovedností, zkušeností, nástrojů a technik k definování, řízení, vizualizaci, měření, kontrole, sledování a zlepšování procesů, které poté mohou být optimalizovány a následně také úspěšné.

2.2 Vlastnosti procesního řízení

Základní vlastnosti procesního řízení lze shrnout do následujících bodů:

- Procesní odpovědnost – odpovědnost je v procesním řízení přesně definována, dodržována a poté snadno zpětně vysledovatelná.
- Eskalační řízení (optimalizace) – sledování workflow v čase a logiky na pozadí umožňuje zkrácení procesních cyklů a tím výrazně celý proces zrychlit.
- Zpracování na základě události – procesní řízení umožňuje dynamické změny pomocí zautomatizovaným reakcím na obchodní události. Stačí pouze upravit procesy a zařadit je zpět do běhu firmy.
- Aplikační integrace – úspěšně namodelované procesy je možné integrovat do informačního systému a tím zefektivnit a zjednodušit práci s těmito informacemi.
- Grafické navrhování a modelování procesů – snadná, rychlá a efektivní implementace procesů a podchycení změn.
- Zprůhlednění organizace – při spolupráci mezi více firmami je potřeba správně definovat vztahy mezi nimi.

2.3 Proces

Definic procesu je velké množství, liší se ovšem pouze úhlem pohledu. Definice procesu podle normy EN ISO 9000:2000 zní:

Definice 2.1 *„Proces je definován jako soubor vzájemně souvisejících nebo vzájemně působících činností, který přeměňuje vstupy na výstupy“.*

Michael Hammer a James Champy, jedni ze zakladatelů BPR (Bussiness Process Reeingeneering), definovali proces takto:

Definice 2.2 *„Proces je soubor činností, který vyžaduje jeden nebo více druhů vstupů a tvoří výstup, který má pro zákazníka hodnotu“.*

Z výše uvedených definic můžeme logicky vyvodit, že proces využívá nějaké zdroje na vstupu, určitým způsobem je transformuje na výstup a skládá se z uspořádaných dílčích činností (kroků) – lze jej tedy dekomponovat na subprocesy.

2.4 Spojité a oddělené procesy

Kategorie procesů se rozděluje na spojité procesy a oddělené procesy. U spojitých procesů, které jsou typickým příkladem fyzických procesů, probíhají spojité změny nepřetržitě. Naopak u jednotlivých (oddělených) procesů, s kterými se setkáváme u počítačových programů, se změny vyskytují v samostatných krocích nazývaných události. Tyto události jsou prokládány s časovými úseky nečinnosti nazývanými stavy.

Jednotlivé úseky spojitých a oddělených procesů:

- Spojité procesy
  - Zahájení
  - Pokračování (zachování)
  - Zastavení
- Oddělené procesy
  - Události
  - Stavy

2.5 Dělení procesů dle různých kritérií

Detailní rozdělení procesů podle Erika Sandewalla, který bere v úvahu proměnné jako čas, změny, posloupnosti a souvislosti procesů:

- Jednotlivé (oddělené) a spojité – v informační technologii je čas rozdělen na jednotlivé body, nazývané „integer time“, který se skládá z celých číslic 0, 1, 2 ... .Spojité procesy mohou být zaokrouhleny snížením časového kroku, čímž stoupá počet bodů, které musí být uloženy a propočítány. Naopak u oddělených procesů, kde hodnoty nabývají definovatelný počet hodnot, je počet bodů na propočítání menší.
- Lineární a rozvětvené – lineární pořadí časových bodů a jejich přidružených událostí tvoří deterministický proces, který lze jednoduše reprezentovat a propočítat. Podmíněné alternativy vytváří další větvení s nedeterministickým zvyšováním možností v budoucnu, které musí být analyzovány.

- Závislé a nezávislé – jestliže jsou procesy nezávislé, změna jednoho nemá efekt na jiné procesy. U závislých procesů, změna jednoho způsobí změny u ostatních.
- Okamžité a zpožděné – pokud změny způsobila nějaká událost, dojde k okamžité změně a tyto změny mohou být zastoupeny nebo simulovány v průběhu stejného časového úseku jako událost. Zpožděný efekt ovšem nezpůsobí pozorovatelnou změnu do některých následujících stavů nebo událostí.
- Postupné a souběžné – v postupných procesech nastane pouze jedna událost v každém okamžiku, naopak v souběžných procesech probíhá více nezávislých událostí paralelně.
- Předvídatelné a nepředvídatelné – předvídatelný proces vyplývá ze skriptu, který specifikuje všechny možné stavy pro každou událost. Nepředvídatelné události vyvolávají změny, které nejsou očekávány podle skriptu.
- Pravidelné a nepravidelné – některé procesy mají vysoce pravděpodobný nebo normální průběh událostí, který může být předpokládaný jako standardní. U nepravidelných procesů jsou některé události stejně pravděpodobné a žádný jednotlivý výsledek nelze považovat za standardní.
- Ploché a hierarchické – u plochých procesů je každá událost popsána stručným seznamem změn, které mohou nastat. U hierarchických procesů, každá událost může být složená z dílčích událostí.
- Časově závislé a nezávislé – nejjednodušší proces k analýze zahrnuje pevný počet časově nezávislých objektů, které nejsou ani vytvořené ani zrušené během procesu. Časově závislé objekty, které mohou být vytvořeny nebo zrušeny během určitého intervalu, vedou ke zpracování s neustále se měnícími objekty.
- Paměťově závislé a nezávislé – u paměťově nezávislých procesů, budoucí průběh událostí závisí pouze na současném stavu. Paměťově závislé procesy udržují informace z předchozích stavů, které mohou mít význam do budoucích stavů.

2.6 Metody pro popis procesů

Abychom mohli analyzovat složité systémy, které jsou propojeny komplikovanými vazbami, jednak ve své vlastní struktuře nebo s okolními subjekty, používáme zjednodušených modelů. U těchto zjednodušených modelů ovšem nesmějí chybět zásadní stavy a funkce, bez kterých by tento model nebyl zcela funkční.

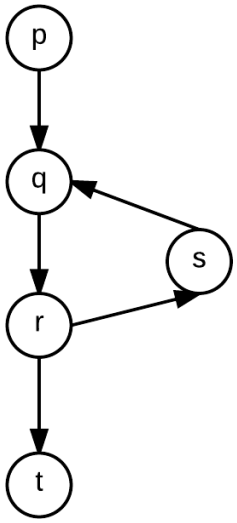
Význam modelu podnikového procesu dále umocňuje fakt, že skrze tyto modely znázorňující procesy ve formě diagramu vede ke zjednodušené komunikaci manažerů a koncových uživatelů, a také pro lepší představu a snadnější pochopení. Model tedy můžeme chápat jako abstraktní obraz reality s cílem ulehčit komunikaci mezi manažery a analytikem nebo developerem.

Model nejčastěji značíme jako procesní diagram, workflow diagram nebo diagram datových toků. Hotový vytvořený model představuje přijatelnou formu pro manažery,

kteří těmto modelům rozumí a rychle se v nich orientují.

Pro modelování využíváme tyto základní druhy diagramů:

- Přechodový diagram - tento diagram se běžně používá pro samostatné procesy. V přechodového diagramu se reprezentují stavy jako kružnice a události znázorňují pomocí šipek, které propojují jednotlivé stavy. Nejjednodušší podobou přechodového diagramu a taktéž široce používaným je konečný automat, který je vidět na *obrázku 1* [1] (písmena *p, q, r, s, t* jsou stavy diagramu).

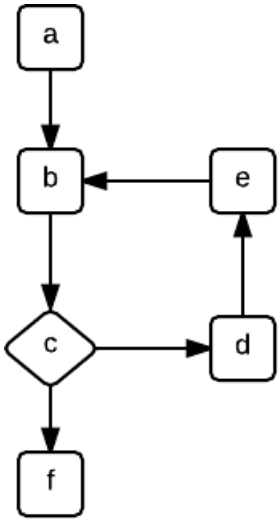


Obrázek 1: Přechodový diagram

- Vývojový diagram - Diagram vhodný pro programování, který doplňuje a zpřesňuje konečné automaty, též stavové diagramy. V tomto diagramu představují bloky (obdélníky) události a diamanty (kosočtverce) reprezentují rozhodnutí, kterým směrem se bude diagram dále ubírat. Na *obrázku 2* můžeme vidět ukázku (písmena *a, b, d, e, f* vyjadřují události, písmeno *c* reprezentuje podmínku - větvení diagramu).

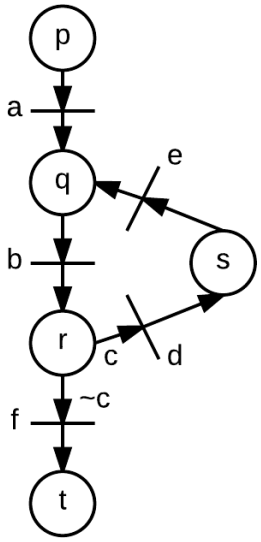
- Petriho sítě - jsou obecný diagram sloužící pro reprezentování souběžných procesů a spojují dohromady vývojový diagram s diagramem přechodovým (stavovým) do jednoho více obecného. Kružnice v Petriho sítích nazvané místa odpovídají stavům v přechodovém diagramu. Mříže nazvané přechody jsou shodné s událostmi ve vývojovém diagramu. U objektově orientovaného návrhu byly Petriho sítě přijaty jako základ pro diagram aktivit v jazyce UML.

[1] Všechny diagramy jsem vytvářel v programu (internetové aplikaci) Lucidchart (https://www.lucidchart.com).



Obrázek 2: Vývojový diagram

Obrázek 3 ukazuje Petriho síť vzniklou z předchozího vývojového a stavového diagramu (písmena *p, q, r, s* a *t* představují stavy diagramu jako u stavového diagramu, písmena *a, b, d, e* a *f* reprezentují události mezi stavy a písmeno *c* vyjadřuje podmínku. Jestliže je podmínka splněna pokračuje se ze stavu *r* do stavu *s*, jestliže podmínka splněna není pokračuje se ze stavu *r* do stavu *t*. Stav *p* je označován jako předpoklad pro událost *a* a stav *q* je nazýván důsledek události *a* a předpoklad pro událost *b*.)



Obrázek 3: Petriho síť

2.7 UML

Unified Modeling Language je prostředek, nástroj, soubor pravidel pro grafickou interpretaci, který se využívá při vývoji systémů. UML je grafický jazyk, který je dnes již standardem pro návrh softwaru. S UML se v současné době setkáme v dokumentacích, odborných materiálech a ulehčuje komunikaci a pochopení mezi návrhářem, vývojářem (developerem) a koncovým zákazníkem. Je také nepostradatelnou součástí při vývoji v týmu, kdy slouží jako komunikační prostředek mezi programátory.

Způsoby využití UML:

- Koncept (náčrt, sketch) - používá se pro jednoduché a rychlé kreslení diagramů, nejčastěji ručně na papír při jednání se zákazníky, nebo na tabuli při komunikaci v týmu během vývoje. Každý diagram je abstraktní, vyjadřuje určitý pohled na systém, přičemž bereme v úvahu právě tento pohled, zbytek systému nebereme v úvahu.
- Detailní návrh (plán) - využívá se pro detailnější plán implementace hlavně pro programátory. Analytik by měl při kreslení návrhu zaznamenat všechny součásti systému pro snadné pochopení programátora tak, aby programátor mohl vytvořit systém bez většího uvažování nad správnou funkčností. Nejčastěji jsou diagramy vytvořeny v CAD nástrojích. Když je systém hotový, slouží dále tento plán a všechny související diagramy jako dokumentace.
- Programovací jazyk - slouží pro vygenerování kódu, který se dále používá jako šablona. Tato šablona se následně používá jako základ pro implementaci, pro základní verzi spustitelného kódu nebo jako výchozí vygenerování skriptů pro databázi.

Nejpoužívanější součástí standardu UML jsou diagramy, které se dělí následovně:

- Diagramy struktury - charakterizují stavbu systému
  - Diagram tříd
  - Diagram komponent
  - Diagram nasazení
  - Diagram balíčků
  - Diagram objektů (instancí)
- Diagramy chování - popisují chování systému
  - Diagram aktivit
  - Diagram užití
  - Stavový diagram

  U diagramů chování můžeme nalézt ještě samostatnou kategorii

- Diagramy interakce - objasňují interakci mezi částmi systému
  - Sekvenční diagram
  - Diagram komunikace
  - Diagram interakcí
  - Diagram časování

Nyní popíši nejčastější diagramy, které jsem použil při návrhu mé aplikace.

2.7.1 Diagram tříd

Diagram tříd, též Class diagram určuje, jak má programátor implementovat jednotlivé třídy a vztahy mezi nimi. Zobrazuje tedy třídy jako typy objektů, atributy a druhy atributů a statické vztahy, které existují mezi třídami. Při vytváření tohoto druhu diagramu musíme myslet na to, že diagram je platformě závislý - například u definování datového typu atributu, který je specifický pro daný programovací jazyk.

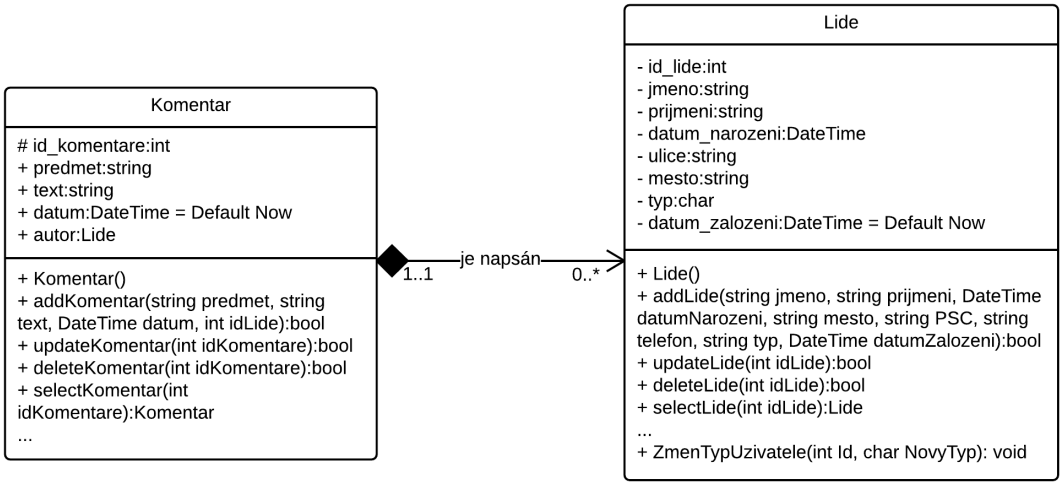
Na *obrázku 4* je vytvořen třídní diagram, ve kterém vidíme třídy *Lidé* a *Komentář*. Tyto třídy mají v horní části název, uprostřed seznam atributů (například u *Komentáře* je to jeho jednoznačně identifikovatelné číslo *ID*, *předmět*, *text* a další) a v poslední části se nachází seznam metod a funkcí (například metoda na úpravu již existujícího komentáře *updateKomentar*).

Vztahy mezi jednotlivými třídami jsou vyjádřeny plnými čárami, kde směr šipky ukazuje závislost mezi těmito třídami. Druhý konec šipky nám říká, zda jde o vztah agregace, kompozice nebo asociace. Vztah agregační se značí prázdným kosočtvercem a říká nám, že daná třída může existovat bez třídy, na kterou má vazbu. Vztah kompozice se značí plným kosočtvercem a říká nám naopak, že třída nemůže existovat sama (jako je to u vztahu na *obrázku 4*, kde třída *Komentář* nemůže existovat bez třídy *Lidé* - autor komentáře). Poslední údaj, který lze vyčíst z *obrázku 4* je, že člověk může vložit více komentářů (0..*), ale opačně jeden komentář je vázán právě k jednomu autorovi tohoto komentáře (1..1).

Atributy jsou značeny ve tvaru *viditelnost název:typ = implicitní hodnota*. Viditelnost určuje, kde bude atribut viditelný a může byt několika typů:

- - soukromý (private) - jen uvnitř třídy
- + veřejný (public) - ze všech tříd
- # chráněný (protected) - uvnitř třídy a potomků dané třídy

Typ atributu závisí na použití daného programovacího jazyka a určuje, jakým způsobem bude atribut uložen v paměti. Implicitní hodnota určuje počáteční hodnotu, kterou má atribut při vytvoření nového objektu dané třídy. Například + *datum:DateTime = Default Now* - veřejná viditelnost, název datum, datový typ DateTime a implicitní hodnota aktuální čas.



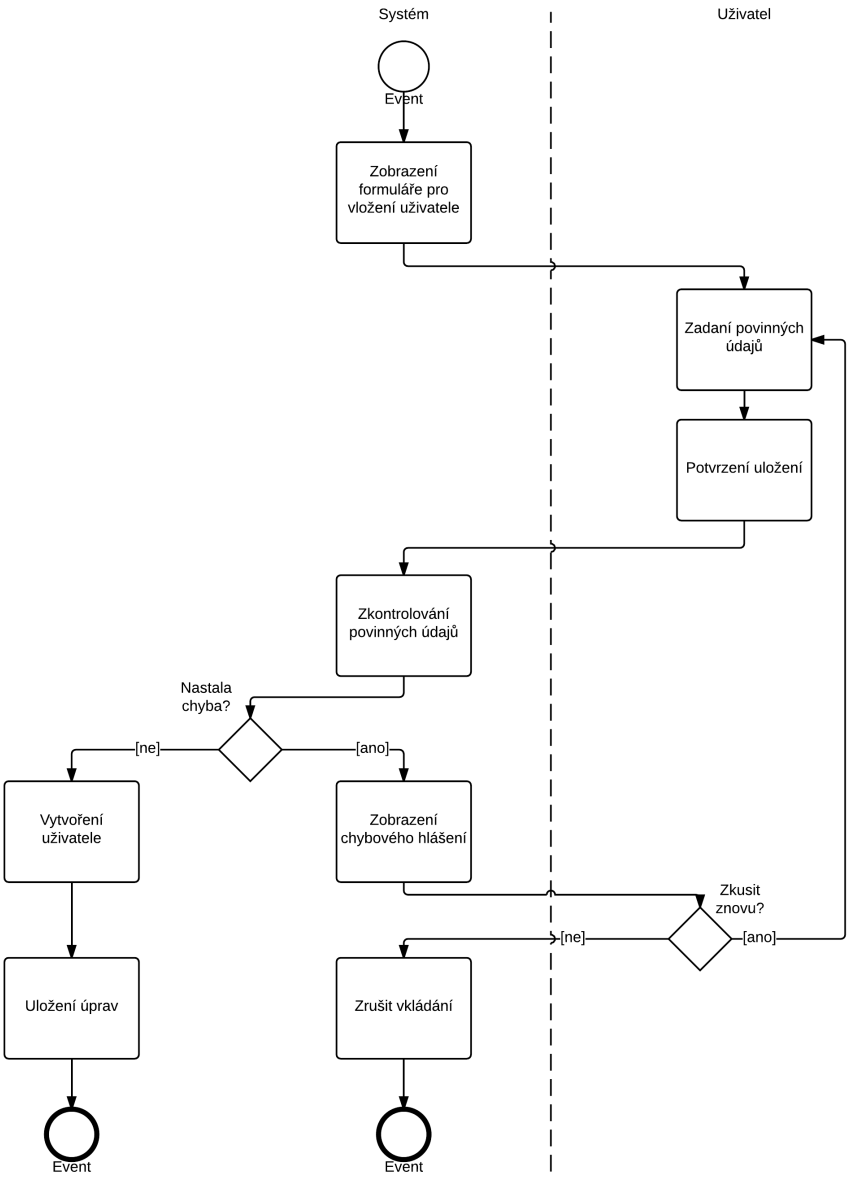
Obrázek 4: Diagram tříd

2.7.2 Diagram aktivit

Diagram aktivit, jinak také Activity diagram se používá pro zobrazení procedurální logiky byznys procesů. Zobrazuje procesy jako aktivity a skládá se z uzlů propojených hranami.

Uzly mohou být rozhodovací - tyto uzly mají jednu vstupní hranu a více hran výstupních. Podle podmínky, která je definována uvnitř uzlu se určí, kterým výstupem z uzlu se bude dále v diagramu pokračovat. Není-li splněna žádná podmínka, pokračuje se výstupem uzlu značeným jako else. Dalším typem uzlu je uzel sloučení, který má naopak více vstupních hran, ale pouze jednu hranu výstupní. Využívá se především pro sjednocení jednotlivých větví rozvětveného diagramu, který byl předtím rozdělen. Dalšími typy uzlu jsou uzly akční, které inicializují nebo provádějí určitou aktivitu. Diagram by měl vždy začínat počátečním uzlem značeným plným tenkým kruhem a končit by měl v koncovém uzlu značeným hrubým kruhem (někdy též kruhem s tečkou).

Pro zpřehlednění je velmi doporučené diagram rozdělit do plaveckých drah nebo zón odpovědnosti podle rolí, kdy se jednotliví aktéři (nebo také aktér a systém) vzájemně střídají. Na *obrázku 5* je znázorněn Activity diagram, který popisuje proces vložení nového uživatele do systému, ve kterém vystupuje jako aktér uživatel a systém.

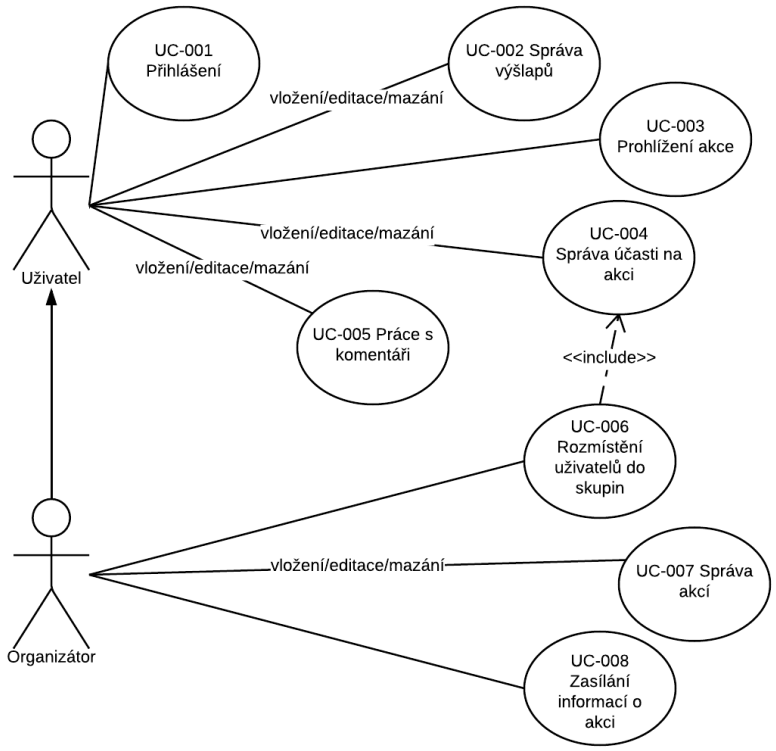


Obrázek 5: Diagram aktivit

2.7.3 Diagram užití

Diagram užití, jinak také Use Case Diagram, definuje a popisuje případy užití systému uživatelem. Zachycuje tedy chování systému z pohledu uživatele, určuje, které typy uživatelů vstupují do systému a jaké činnosti mohou vykonávat v rámci popisovaného sys-

tému. Diagram užití bývá zpravidla doplněn o textovou podobu (scénář případu užití), která dále zpřesňuje každý jednotlivý případ užití.



Obrázek 6: Diagram užití

Prvky diagramu:

- Aktér (účastník) - značí se figurkou a popisuje uživatele vstupující do vztahu s procesy. Uživatel nemusí být pouze fyzická osoba, ale může se jednat o jiný systém nebo hardwarové zařízení. Název uživatele vystihuje jeho roli v celém systému. Jednotliví účastníci mohou hierarchicky dědit jednotlivé případy užití od svých předků.
- Případ užití - značí se oválně a popisuje část funkcionality systému, kterou používají aktéři. Obvykle bývá zpřesňován o scénář případu užití, který charakterizuje posloupnost kroků prováděných za určitým cílem. Typy vztahu používané mezi případu užití:
  - Include - případ užití může obsahovat jiný případ, který může být využíván na více místech

- Extend - případ užití může rozšiřovat jiný případ, který musí být splněn
- Generalizace (specializace) - tímto typem můžeme také znázornit hierarchii u aktérů (tedy jejich předka či potomka)

Na *obrázku 6* je znázorněn diagram užití znázorňující správu a účasti na nějakých akcích. V diagramu vystupují dva aktéři a to obyčejný uživatel, kterého rozšiřuje aktér organizátor (je mezi nimi tedy vazba type generalizace). Organizátor může tedy využívat všechny funkčnosti systému jako uživatel a navíc má několik funkcí, ke kterým nemá uživatel přístup (například správa akcí, zasílání informací). Případ užití UC-006 Rozmístění uživatelů do skupin navíc využívá případ užití UC-004 Správa účasti na akci (vazba typu include). Když organizátor rozdělí jednotlivé uživatelé do skupin, promítne se tato změna i uživatelům.

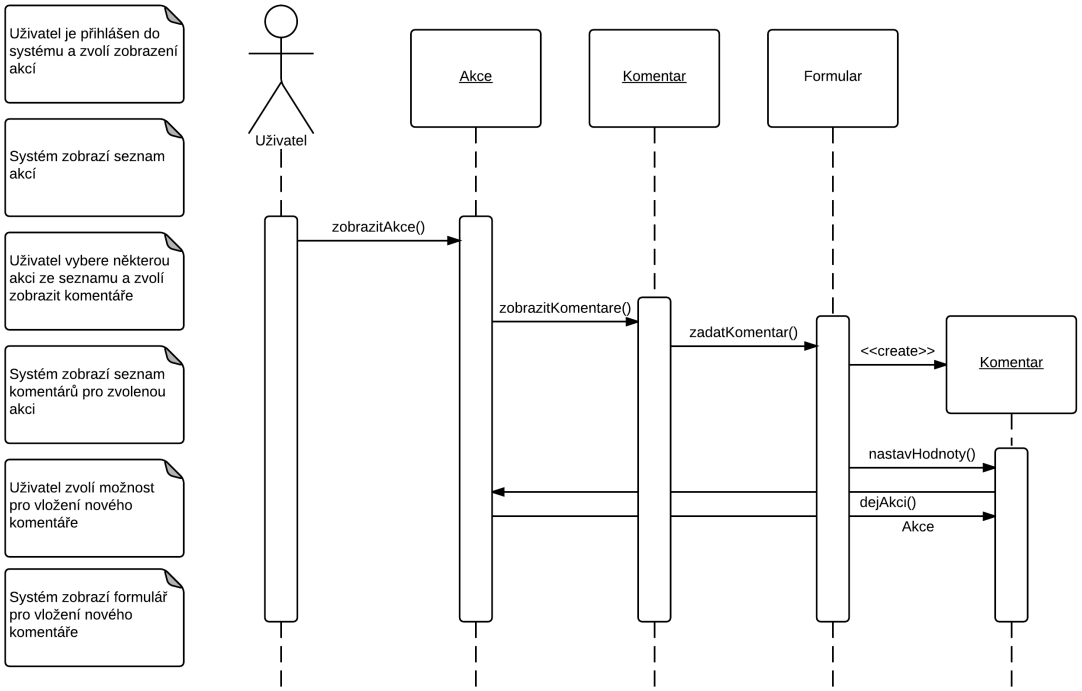
2.7.4 Sekvenční diagram

Dalším diagramem používaným při návrhu aplikace je sekvenční diagram zobrazující posloupnost zpracování zasílaných zpráv mezi objekty. Diagram nejčastěji zachycuje chování a komunikaci několika objektů v rámci jednoho konkrétního případu užití.

Prvky diagramu:

- Klasifikátory - znázorňují objekty, třídy nebo aktéry komunikující spolu v rámci diagramu
- Zprávy - zprávy, procedury, funkce, kterými komunikují v sekvenčním diagramu mezi sebou klasifikátory
- Čára života - reprezentuje jak dlouho a jakým způsobem se instance klasifikátoru účastní dané komunikace. Čára života vede z každého klasifikátoru.
- Iterace - diagram může obsahovat i iterace, podle podmínky se určí, která část bude následovat

Na *obrázku 7* je znázorněn neúplný sekvenční diagram pro vložení nového komentáře. Vlevo v diagramu jsou zobrazeny poznámky k jednotlivým krokům, které je dále zpřesňují, například že uživatel musí být přihlášen do systému. V diagramu vystupuje jeden aktér a to *uživatel*, který chce vložit nový komentář. Dále zde vystupují objekty *Akce, Komentář* a *Formulář*, a vzniká nová instance objektu *Komentář*, která se postupně naplní daty od uživatele a systému. Čáry života zde začínají vždy při požadavku nějakého klasifikátoru na jiný.



Obrázek 7: Sekvenční diagram

3 Popis vývojového prostředí Android Studio BETA použitého při vývoji a popis práce s tímto prostředím

3.1 Vývojové prostření pro vývoj Android aplikací

V současné době máme na výběr dvě hlavní možnosti pro vývoj Android aplikací a to:

- Eclipse s ADT
- Android Studio

Pro mou aplikaci jsem zvolil Android Studio kvůli níže uvedeným důvodům.

3.2 Eclipse s ADT

Původní vývojový nástroj Eclipse IDE s pluginem Android Developer Tools byl základním a doporučeným způsobem jak vytvářet Android aplikace. Instalace je velice komplikovaná, je potřeba nainstalovat JDK (Java Development Kit), který slouží jako základní nástroj pro práci s Javou, ať už programujeme desktopové nebo mobilní aplikace.

Dalším krokem je nainstalování SDK (Software Development Kit) pro Android, jedná se v podstatě o balíček s různými nástroji pro práci s danou platformou, které využívá Eclipse. Balíček obsahuje například nástroje pro úpravu grafických prvků, testování a optimalizaci vzhledu obrazovek.

Abychom mohli všechny předchozí funkce využívat a pracovat s nimi, musíme do prostředí Eclipse přidat plugin ADT (Android Development Tools), který nám do IDE přidá nabídky týkající se Androidu.

Poslední částí je stažení konkrétní verze platformy (operačního systému Android) a také emulátoru androidu. Emulátor funguje jako náhrada běžného telefonu se systémem Android a používá se pro testování a ladění vyvíjené aplikace. Emulátor běží na odlišné platformě než běžný stolní počítač a proto je jeho běh a reakce celkově pomalejší než u reálného zařízení.

V dnešní době se vyvíjet s IDE Eclipse dá, ovšem postupně se od něj upouští a přechází se na Android Studio.

3.3 Android Studio

Nové vývojové prostředí od společnosti Google, které je dnes již upřednostňováno. Plně připraveno pro vývoj - vše v jednom, jednoduchá instalace, nepotřebujeme doinstalovávat a konfigurovat žádné pluginy. Jediné, co je potřeba pro vývoj je JDK. Android Studio je založeno na IntelliJ IDEA, pokud tedy již vyvíjíme pod nějaký programovacím jazykem, například PHP a používáme nástroj PHPStorm, je velice snadné zvyknout si na toto IDE. Umožňuje snadnou práci s kódem (navigace v kódu, nápověda, refaktoring, našeptávání a další).

Pro vývoj mé aplikace jsem zvolil ještě BETA verzi tohoto nástroje, dnes však už můžeme používat verzi 1.2, tedy oficiální.

3.4 Lokalizace aplikací

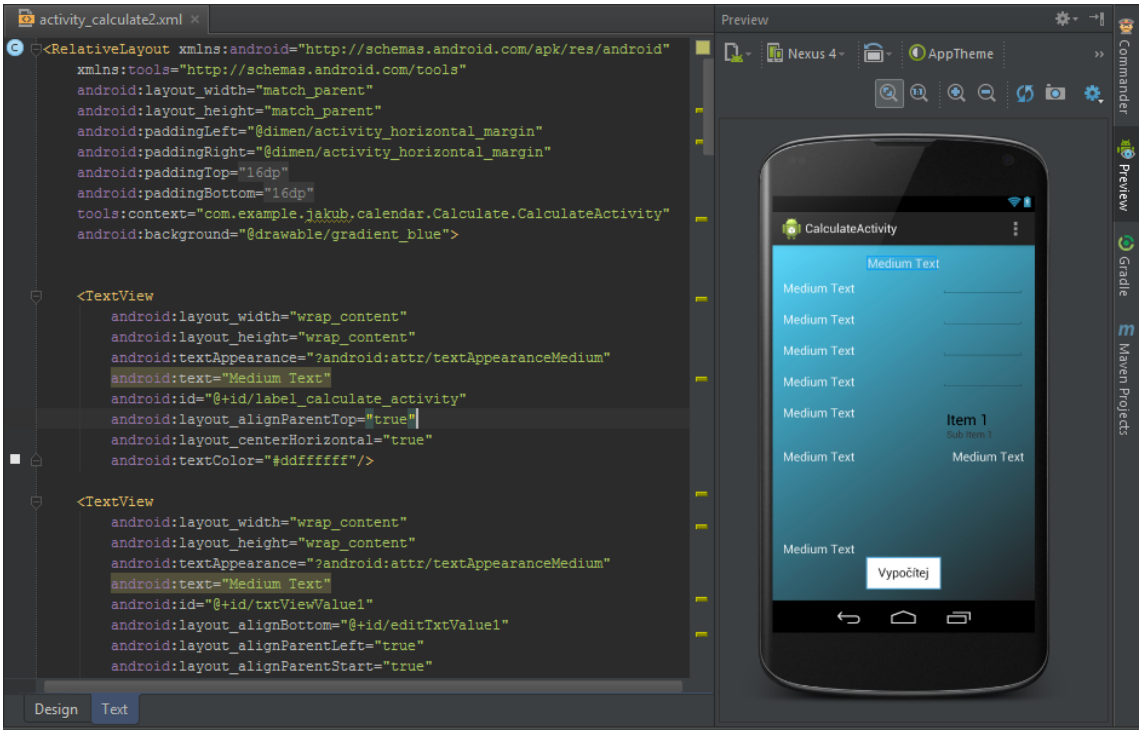
Android Studio podporuje více jazyčný přístup k vývoji aplikací. Pokud chceme přizpůsobit aplikaci i jiným jazykům, ukládáme stringy v souboru *strings.xml*, která se nachází v adresáři *resources*. Poté stačí zvolit funkci *Add transaltion*, vybrat požadovaný jazyk a zobrazí se nám přehledně v tabulce jednotlivé stringy a jejich editovatelné překlady.

3.5 Design

Android Studio umožňuje navrhovat vzhled aplikací - jednotlivých ploch (zde se plochy nazývají layout) ve dvou režimech: Design mód a textový mód.

V design módu uchopíme některý z možných prvků (například *Button* - tlačítko) a funkcí Drag and Drop, tedy přetažením kurzorem myši, vložíme do layoutu. Android Studio je v tomto režimu daleko přesnější než výše zmíněné prostředí Eclipse. Při použití relativního rozložení (Relative layout) se jednotlivé prvky přichycují k jiným nebo se umisťují v určité poloze vzhledem k obrazovce (například doprostřed obrazovky).

Pokud využijeme možnost tvořit design pomocí textového módu - píšeme XML kód. Android Studio okamžitě reaguje na změny a zobrazuje náhled ve vedlejším okně aplikace v požadovaném rozlišení. Díky této vlastnosti se nemusíme neustále přepínat mezi jednotlivými módy.



Obrázek 8: Textový režim návrhu vzhledu aplikace s náhledem

Při navrhování designu je velmi užitečnou vlastností Android Studia funkce *Preview All Screen Sizes*, při které se zobrazí náhledy všech rozlišení zařízení najednou. Můžeme tedy vidět, jak se daný návrh chová na různých zařízeních s různým rozlišením.

3.6 Emulátor

Součástí Android SDK je virtual mobile device emulátor, tedy virtuální mobilní zařízení, které je spuštěno na počítači. Emulátor umožňuje vytvářet, vyvíjet a testovat aplikace pro zařízení s OS Android bez nutnosti použít fyzické zařízení.

Emulátor napodobuje všechny hardwarové i softwarové funkce pro klasické mobilní zařízení s výjimkou uskutečnění volání. Emulátor poskytuje navigační a ovládací tlačítka, kterými můžeme pomocí stisku tlačítka na klávesnici nebo myši generovat události pro naši aplikaci. Zobrazuje také okno, ve kterém se objevuje vyvíjená aplikace i s dalšími aktivními aplikacemi systému Android.

Emulátor využívá Android Virtual Device (AVD) konfigurace abychom mohli aplikaci snadněji modelovat a testovat. AVD umožňuje definovat určité hardwarové funkce napodobovaného telefonu a umožní tak vytvořit mnoho konfigurací pro testování různých Android platforem a hardwarových výbav. Při běhu aplikace na emulátoru, může tato aplikace využít služby dostupné systému Android, například vyvolat jiné aplikace, může přistupovat k síti, přehrávat videa a zvuky, ukládat nebo číst data, upozornit uživatele (notifikace), měnit vzhled telefonu. V emulátoru můžeme také využít konzole k ladícím účelům nebo simulovat přerušení běžící aplikace - přijetí SMS zprávy nebo hovoru.

Android emulátor podporuje několik hardwarových "features", které jsou k dispozici na mobilních zařízeních:

- Procesory ARMv5, ARMv7 nebo x86 CPU (Central Processing Unit)
- 16-bit LCD (Liquid Crystal Display)
- Jednu nebo více klávesnic (klávesnice Qwerty a obdobné Dpad nebo telefonní tlačítka)
- Zvukový čip s vstupními a výstupními funkcemi
- Flash paměti (emulované přes diskové image soubory na počítači)
- GSM modem včetně simulované SIM (Subscriber Identity Module) karty
- Kamera, využívající webové kamery připojené k počítači
- Senzory jako akcelerometr, používající data na základě údajů z připojeného USB zařízení

Android Studio obsahuje již použitelné emulátory pro NEXUS 4, 7 a 10, NEXUS S, NEXUS ONE, GALAXY NEXUS a další bezejmenné zařízení s různými druhy rozlišení. Emulátor lze přizpůsobit přesným požadavkům - například verze Android API, velikost RAM (Random-Access Memory), úložiště a kapacita SD karty, přední a zadní kameru.

Emulátory jsou spolehlivé, ale velmi pomalé i na PC s výkonným hardwarem a není to problém pouze Android Studia. I z tohoto důvodu jsem při vývoji mé aplikace používal zařízení přímo připojené přes USB port. Počáteční dlouhá doba nahrávání aplikace do telefonu (delší než u emulátoru) je vykompenzována rychlým, svižným a zcela nativním během.

Obrázek 9: Ukázka emulátoru v Android Studiu

3.7 Ladění aplikace

Jak již bylo zmíněno výše, můžeme ladit aplikaci buď'to v emulátoru, nebo využit mobilní zařízení připojené přes USB port k počítači. Abychom mohli ladit aplikaci v telefonu (případně tabletu), musíme nejdříve povolit ladění přes USB. U verze Androidu 4.1.x nalezneme tuto možnost v sekci *Nastavení - Možnosti pro vývojáře - Ladění USB*.

U androidu 4.2 nebo ve vyšších verzích systému je to nepatrně složitější. Z bezpečnostních důvodů je režim ladění USB určen výlučně vývojářům, protože jeho povolením se může méně zkušený uživatel snadno dostat do potíží. Za tímto účelem si u nových verzí Androidu musíme položku *Pro vývojáře* odemknout. V menu *nastavení* vybereme položku *Informace o tabletu - Verze jádra*. Na tuto položku je potřeba 7x poklepat a pak už stačí jen v zpřístupněné nabídce *Pro vývojáře* zaškrtnout možnost *Ladění USB*.

Dobrou radou je vypnout automatické zhasínání obrazovky během ladění. V menu *Nastavení - Možnosti pro vývojáře* zvolíme možnost *Nevypínat obrazovku*. Pro testování aplikací využívající GPS modul je možné povolit funkci *Simulované polohy*. V internetovém obchodě GooglePlay nalezneme několik aplikací (například Fake GPS), které umožňují

zvolit na mapě polohu a tuto polohu bude poté GPS hlásit jako výchozí.

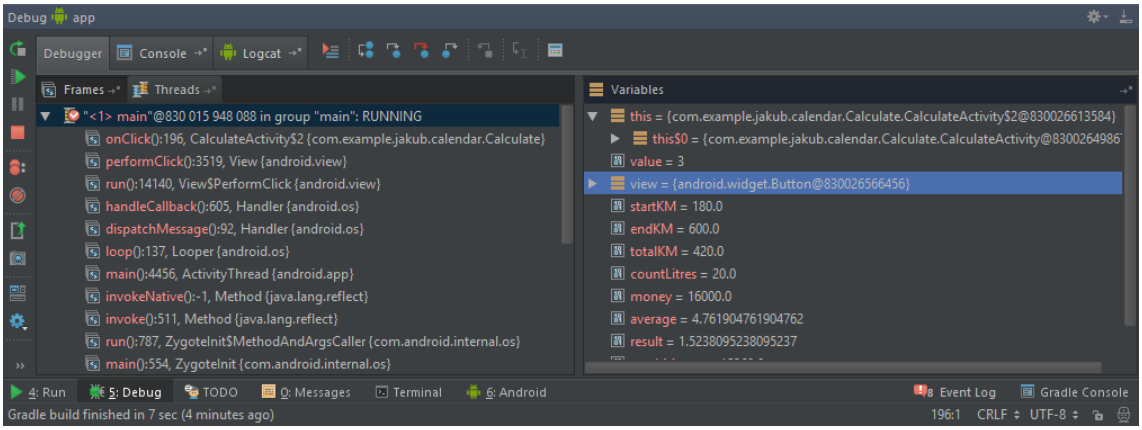
Android Studio umožňuje při ladění využít tyto funkce:

- Zobrazení záznamu logu systému
- Volbu zařízení, které bude použito pro ladění aplikace
- Nastavit Breakpointy v kódu
- Sledovat proměnné a vyhodnocovat výrazy za běhu
- Zachytit snímky a videa z aplikace

Režim ladění - jinak taky Debug Mode spustíme v Android Studio kliknutím na *Debug* v paletě funkcí. Zvolíme bud'to připojené mobilní zařízení, nebo virtuální zařízení - emulátor. Android Studio otevře nástroj pro ladění, ve kterém jsou zobrazena vlákna a proměnné aplikace v záložce *Debugger*, stav zařízení v záložce *Console* a záznam logu systému v záložce *Logcat*.

Během ladění máme k dispozici klasické funkce, jaké známe z jiných vývojových prostředí pro posun v aplikaci vpřed, vstup do funkce nebo výstup z funkce, které využívají Breakpointy. Breakpointy umožňují pozastavit běh aplikace v příslušném bodě, respektive na zvoleném řádku v kódu a prozkoumat tak hodnoty proměnných, vyhodnocení výrazu a dále pokračovat řádek po řádku. Používání breakpointů tedy pomáhá nalézt chyby, které není snadné najít v kódu.

Android Studio umožňuje také výpis log zpráv ze zdrojového kódu. Abychom mohli využít tuto vlastnost, musíme použít třídu Log. Log zprávy pomáhají lépe porozumět průběh vykonávání shromažd'ováním výstupu při interakci s aplikací. Log zprávy také mohou ukázat část aplikace, která způsobuje nečekaný pád. Tyto log zprávy si můžeme zobrazit v záložce Logcat a jednoduše filtrovat pouze ty, které nás aktuálně zajímají.

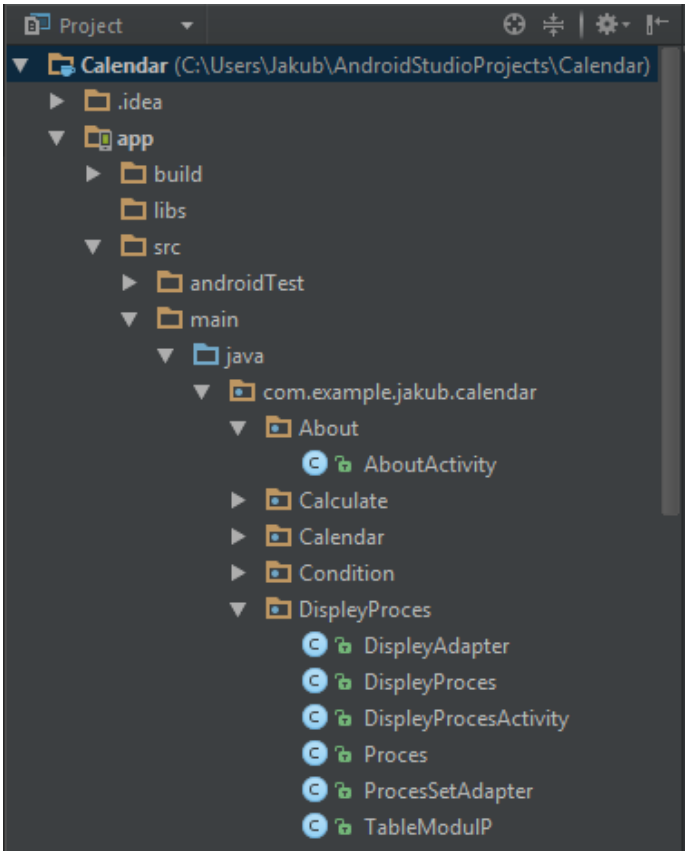


Obrázek 10: Debug Mode v Android Studiu

3.8 Struktura projektu

Struktura projektu, který byl vytvořen v Eclipse je odlišná od projektu založeném v Android Studiu. Každá instance z Android Studia obsahuje projekt s jedním nebo více aplikačních modulů. Každý adresář aplikačního modulu obsahuje kompletní zdrojové soubory pro tento modul, včetně *src/main* a *src/androidTest* adresářů, zdroje pro aplikaci v adresáři *res* (například layouty, tedy design obrazovky aplikace), build soubor a soubor *Android Manifest*. Nejčastější změny zdrojových kódů se uskutečňují v adresáři *src/main* v každém modulu. Soubor *gradle.build* slouží pro specifikaci aplikace a adresář *src/androidTest* slouží pro testování aplikace.

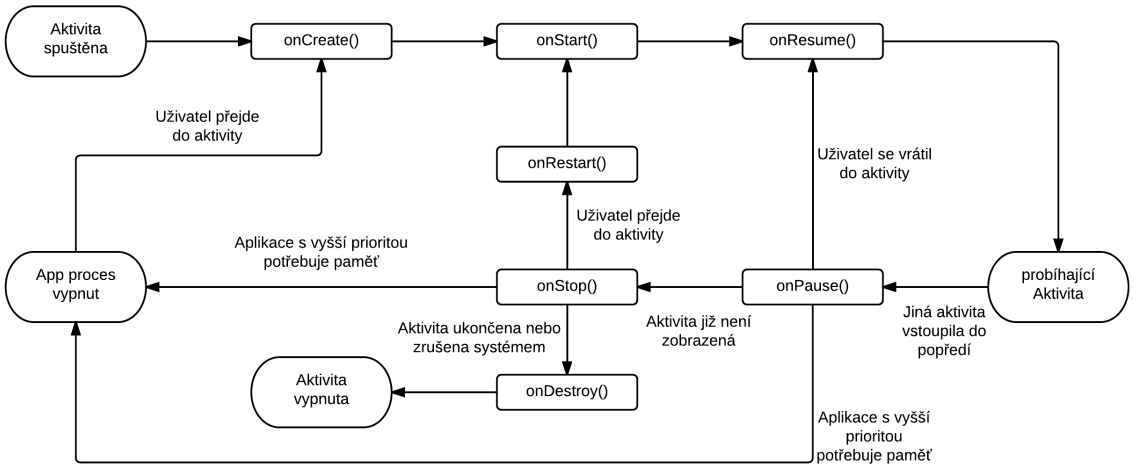
V adresáři *src/main* se nachází adresář *java*, obsahující další podadresář, ve kterém se nacházejí už jednotlivé zdrojové kódy aplikace. Je vhodné tyto kódy tématicky a logicky rozdělit do jednotlivých celků - adresářů, pro přehlednost a rychlou orientaci v projektu. Programování pro Android má jednu odlišnost a to, že se skládá z Aktivit. V podstatě je doporučeno mít pro každou obrazovku aplikace - tedy layout, právě jednu aktivitu. Jinak se projekt jako u jiných programovacích jazyků dělí do tříd.



Obrázek 11: Struktura projektu v Android Studiu

Aktivita je v podstatě stavební kámen celého programu, bez kterého by aplikace nešla spustit. Aktivita obstarává téměř všechny interakce s uživatelem, vytváří tedy okno, do kterého můžeme umisťovat prvky uživatelského rozhraní. Dalo by se tedy říct, že aktivita je prezenční vrstva aplikace. Aktivita musí vždy obsahovat dvě metody a to:

- *onCreate* - metoda, ve které inicializujeme naši aktivitu. Další důležitou funkcí je volání *setContentView* s parametrem layoutu, který definuje naše uživatelské rozhraní.
- *onPause* - metoda, která je zavolána v případě opuštění spuštěné aktivity. Všechny změny provedené uživatelem by měly být v tomto okamžiku uloženy (obvykle pomocí *ContentProvidera*).



Obrázek 12: Životní cyklus aktivity

3.9 Android Manifest

Každá aplikace vyvíjená pro Android zařízení musí obsahovat soubor *AndroidManifest.xml*. Tento soubor obsahuje základní informace o aplikaci pro systém Android, které musí být dostupné před spuštěním kódu aplikace. Základní funkce AndroidManifestu:

- Pojmenovává Java balíček aplikace. Název tohoto balíčku je unikátní a slouží tedy i jako identifikátor aplikace.
- Popisuje komponenty, z kterých se aplikace skládá - aktivity, služby (service), broadcast receivers a content providers. Vždy při vytvoření nové aktivity nesmíme tedy zapomenout zaregistrovat danou aktivitu do tohoto souboru. Tyto deklarace zajišťují, že systém o těchto komponentách ví a určí, za jakých podmínek mohou být spuštěny.
- Určuje, které procesy bude aplikace zpracovávat.

- Deklaruje, které oprávnění musí aplikace mít, aby mohla přistupovat ke chráněné části API a komunikovat s jinými aplikacemi.
- Dále deklaruje oprávnění, které jsou vyžadována pro interakci s komponenty aplikace.
- Uvádí seznam *Instrumentation* tříd, které poskytují profilování a další informace, které jsou v Android Manifestu pouze během vyvíjení a testování aplikace.
- Uvádí minimální verzi Android API, která je aplikací vyžadována.
- Obsahuje seznam knihoven, které aplikace využívá.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.jakub.calendar" >

    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WRITE_CALENDAR" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/name_main_activity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        ...

        <service android:name=".MyAlarmService" android:enabled="true" />

        <receiver android:name=".MyReceiver"/>
    </application>

</manifest>
```

Výpis 1: Ukázka struktury AndroidManifest.xml

Jak lze vidět z ukázky kódu 1, pro mou aplikaci je potřeba několik povolení ze strany uživatele, a to pro čtení a zápis z kalendáře, čtení a zápis do externí paměti. Je zde vidět cesta k obrázku pro ikonu aplikace, název a téma. Další část obsahuje aktivity (pro přehlednost jsem uvedl pouze hlavní). Následující informace definují službu a receiver používané v mé aplikaci.

4 Analýza a návrh aplikace pro mobilní systém Android 4

4.1 Zadání

Cílem práce je vytvořit mobilní aplikaci pro podporu procesů specifické profese. Profesí, kterou se budu dále zabývat je práce taxikáře, ale už od začátku návrhu jsem aplikaci plánoval tak, aby ji bylo možné lehce převést pro jinou profesi. Nevyhnul jsem se ovšem některým výpočtům, které jsou příznačné jen pro určité druhy povolání. Ty lze však snadno a celkem rychle nahradit jinými.

Činnost taxikáře se odehrává v podstatě pouze v automobilu, a proto je vhodné, aby k ní měl taxikář snadný přístup - využití "chytrého"mobilního telefonu. Slovem chytrého myslím telefony s novými operačními systémy, například Android, iOS nebo Windows Phone. Jelikož podíl na trhu je v jednoznačný prospěch Androidu (přes 76,6% za čtvrté čtvrtletí roku 2014 [2]) a pořízení telefonu s tímto OS není nijak těžké, navíc není v dnešní době ani drahé, rozhodl jsem se vyvíjet právě pro tuto platformu. Dalším kladem je, že vyvíjení aplikací pro OS Android je zdarma - již zmíněné vývojové prostředí Android Studio od společnosti Google je zdarma (platí se pouze registrace účtu do aplikace Google Play).

Dalším důležitým aspektem je verze OS Android. Při bližším zkoumání aktuálního podílu na trhu jsem zjistil, že v podstatě nemá velký význam zabývat se vývojem pod verzi Androidu 4 (tedy 4.0.3), jelikož telefon s verzí pod 4.0.3 používá méně než 7,5% [3] (data z 2 března 2015) uživatelů tohoto OS.

4.2 Specifikace požadavků

Dalším důležitým krokem při vývoji je ujasnit si požadavky, které jsou kladeny na aplikaci. Z předchozích odstavců je jasné, že aplikace bude fungovat na mobilním zařízení, které musí mít každý taxikář při ruce kvůli telefonům ze strany zákazníků. Dalším důležitým faktorem je, aby aplikace fungovala snadno a intuitivně bez větších problémů.

Základní požadavky:

- Aby si mohl taxikář lépe plánovat aktivity do budoucna, měl by výsledný program spolupracovat s kalendářem (nejlépe tedy kalendářem na Android telefonu, tedy Google Calendar).
- Program by měl upozornit na právě probíhající procesy uživatele tak, aby na ně mohl snadno a rychle reagovat.
- Aplikace by měla obsahovat základní procesy, které profese taxikáře obnáší. Tyto procesy musí být lehce upravitelné a použitelné do budoucna.
- Rovněž musí aplikace umět nové procesy vytvářet a ukládat do paměti.

[2] Zdroj: http://www.idc.com/prodserv/smartphone-os-market-share.jsp (25. 3. 2015) - Smartphone OS Market Share, Q4 2014

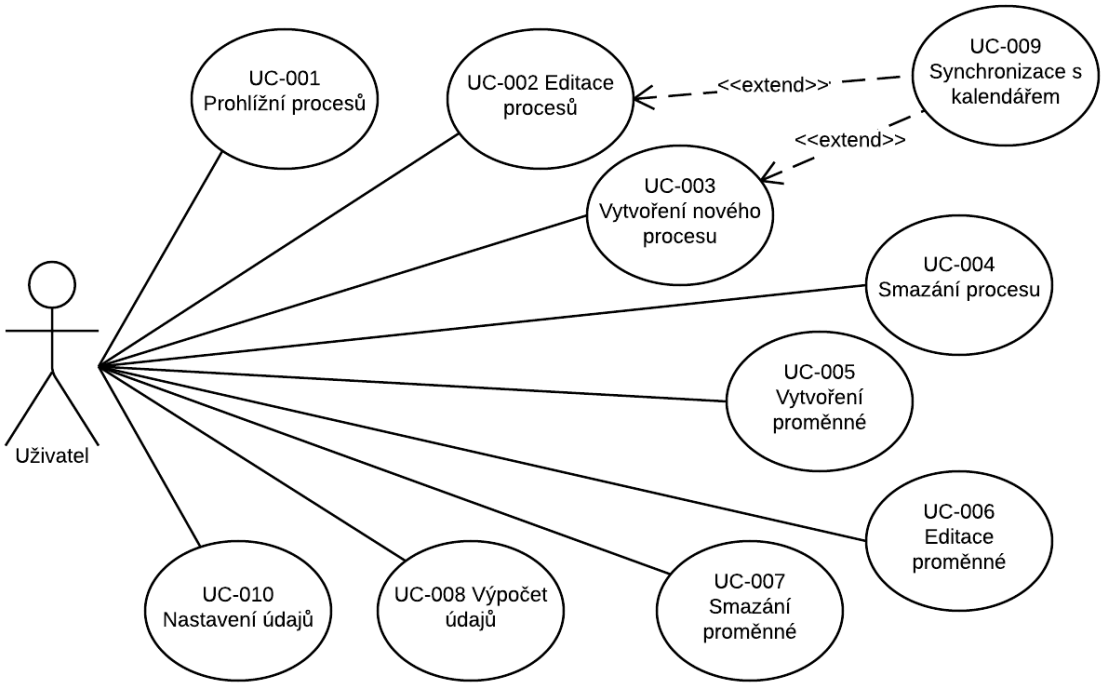
[3] Zdroj: https://developer.android.com/about/dashboards/index.html (25. 3. 2015) - Platform Versions

- Aplikace by měla obsahovat základní výpočty specifické pro daný obor, které počítají s proměnnými rovněž důležitými pro danou profesi.
- Proměnné, které jsou také uloženy v paměti, by měly jít editovat a také přidávat nové.

4.3 Model případu užití

Abychom si přesně vymezili funkčnosti aplikace, je vhodné vytvořit diagram případu užití. Tento diagram jsem popsal přesněji výše. Diagram je vhodný i pro komunikaci se zákazníkem, případně pro konzultaci v týmu.

Jednotlivé možnosti užití jsou popsány v zadání této práce. Umožní taxikáři lépe plánovat jednotlivé procesy a tím ulehčit každodenní vykonávání této profese. Aplikace by měla umožnit komplexní organizaci činností a být podporou pro toto povolání.



Obrázek 13: Diagram případu užití aplikace

Na *obrázku 13* je vytvořený model případu užití pro navrhovanou aplikaci. Co se týká aktérů, tak s aplikací by měli pracovat pouze uživatelé - taxikáři. Jednotlivé případy užití pokrývají povinné body, které musí aplikace obsahovat.

Uživatel si bude moct prohlédnout již existující procesy, které jsou uloženy v aplikaci jako výchozí (případ užití UC-001). Tyto a vlastně vytvořené procesy (UC-003) může uživatel libovolně editovat - například změnu data události (UC-002). Další funkcí, kterou může taxikář používat při vytváření nebo editaci procesů je ukládání/editování/mazání

události v kalendáři (UC-009). Vytvořené procesy by měly jít jednoduše smazat z aplikace (UC-004).

Další důležitou vlastností aplikace jsou výpočty různých údajů (UC-008), které většinou počítají s proměnnými uloženými v aplikaci. Uživatel může proměnné vytvářet (UC-005), modifikovat své nebo již existující proměnné (UC-006) a samozřejmě také smazat z aplikace (UC-007).

Aplikace by měla mít jednoduché nastavení pro synchronizaci s kalendářem a dalšími potřebnými údaji pro fungovaní (UC-010).

4.4 Model tříd

Pro detailnější návrh aplikace slouží třídní diagram. Kvůli zpřehlednění v něm uvedu jen nejdůležitější třídy v aplikaci - jádro aplikace, které se skládá o vytváření, modifikaci a mazání procesu a také o jeho vykreslení. *Obrázek 14* zobrazuje tedy neúplný třídní diagram (vynechal jsem třídy, které slouží například pro vkládání informací do kalendáře nebo třídy, které se starají o provedení výpočtů). Rovněž jsem zkrátil výpis atributů a funkcí. V následujících odstavcích popíši funkčnosti jednotlivých tříd.

4.4.1 Třída Event

Aplikace v podstatě stojí na dvou základních kamenech a to třídě *Event* a třídě *TableModul*. Třída *Event* uchovává všechny důležité údaje o jednotlivých událostech, například jméno, popis, datum začátku a konce události, a další atributy. Podle atributu *calendarSave* se buď'to událost synchronizuje s kalendářem, nebo ne. K synchronizaci slouží položka *calendarId*, což je jednoznačný identifikátor události v rámci daného kalendáře. Abychom uložili stav třídy, použijeme druhou zmíněnou třídu *TableModul*.

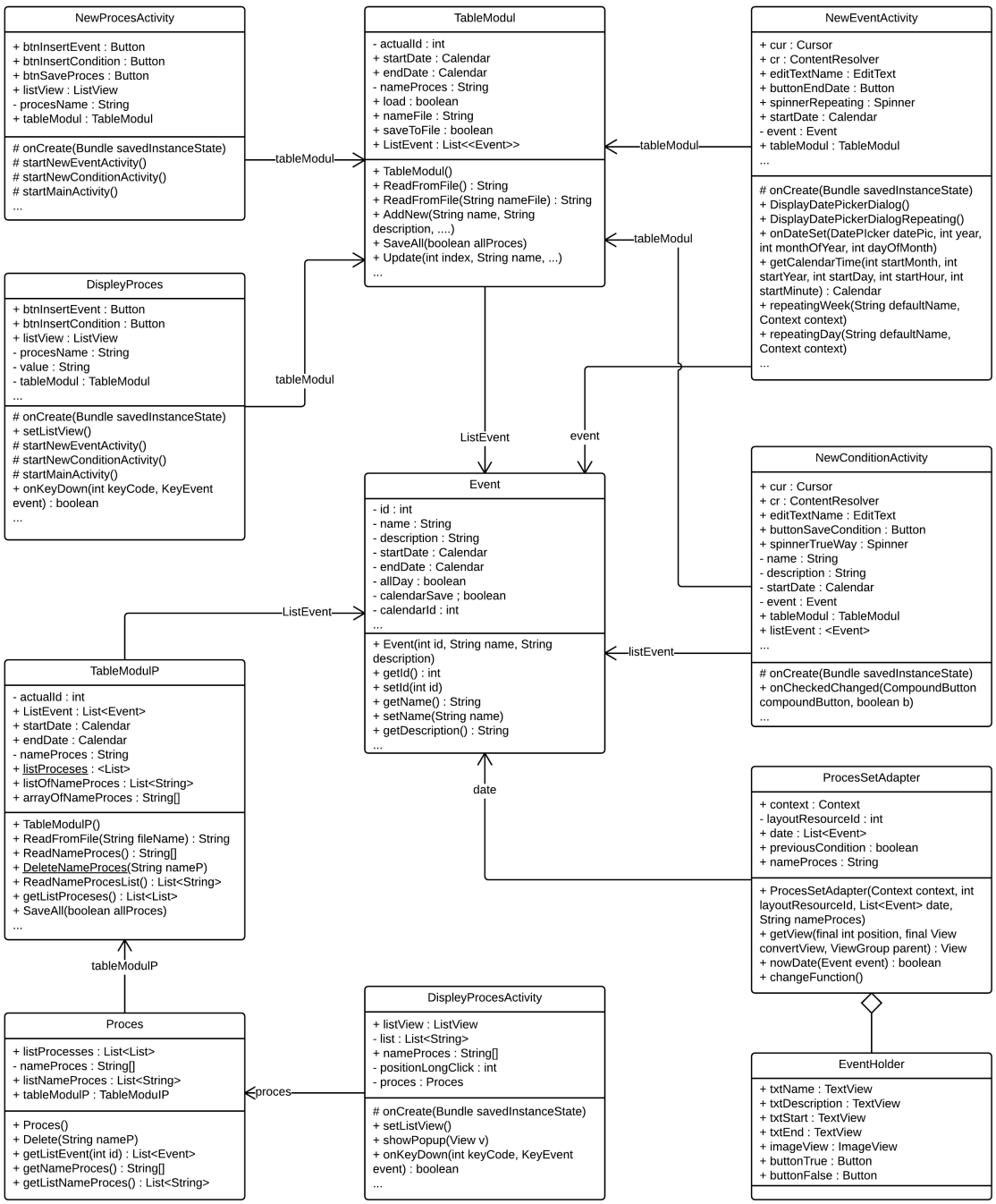
4.4.2 Třída TableModul

Třída *TableModul* se stará o načítání a ukládání důležitých informací z/do telefonu. K tomuto účelu slouží funkce *ReadFromFile()* respektive *SaveAll(boolean allProces)*, které pracují s objektem typu *JSONString*. Tato třída obsahuje také seznam událostí *(ListEvent)*, do kterého můžeme při modelování nového procesu přidávat nové události, tedy objekty typu *Event*. Tento seznam můžeme také modifikovat, tedy upravovat jednotlivé jeho položky - objekty.

Třída *TableModulP*, tedy TableModul proces pracuje obdobně, ale slouží pouze pro zobrazení procesu.

4.4.3 Aktivita NewProcesActivity

Při uložení celého námi vytvořeného procesu se spustí třída - tedy přesněji aktivita, *NewProcesActivity*, která obsahuje atribut *procesName* (název celého procesu) a odkaz na objekt *TableModul* (ten již obsahuje celý namodelovaný proces). Aktivita *NewProcesActivity*



Obrázek 14: Třídní diagram - jádro aplikace

obsahuje metody pro spuštění jiných aktivit, například funkci *NewMainActivity()*. Na podobném principu funguje i aktivita *DisplayProces*

V diagramu tříd zobrazuji i aktivity, které jsem již výše popsal a které jsou v podstatě také třídy, bez kterých by aplikace nefungovala.

4.4.4 Aktivita NewEventActivity

Tato aktivita slouží v podstatě jako prezenční vrstva aplikace při vytváření nové události. Uživatel zadává všechny potřebné údaje pro danou událost:

- název a popis - *EditText*,
- počáteční a konečné datum - *DatePicker*,
- počáteční a konečný čas - *TimePicker*,
- *Checkbox*, který rozhoduje o ukládání do kalendáře a
- opakování události - *Spinner* (volby: neopakovat, opakovat denně, týdně nebo měsíčně)

Při ukládání vzniká nový objekt typu *Event*, který se také vkládá do seznamu objektů - atribut *TableModul*. Atribut *Cursor* slouží pro vkládání dat do kalendáře.

4.4.5 Aktivita NewConditionActivity

Aktivita *NewConditionActivity* slouží obdobně jako předchozí aktivita, ovšem pro vytváření podmínky. Ta posléze slouží pro větvení procesu, proto je nutné správně zvolit, co se má stát při splnění podmínky, respektive nesplnění. Aktivita obsahuje tyto povinné atributy:

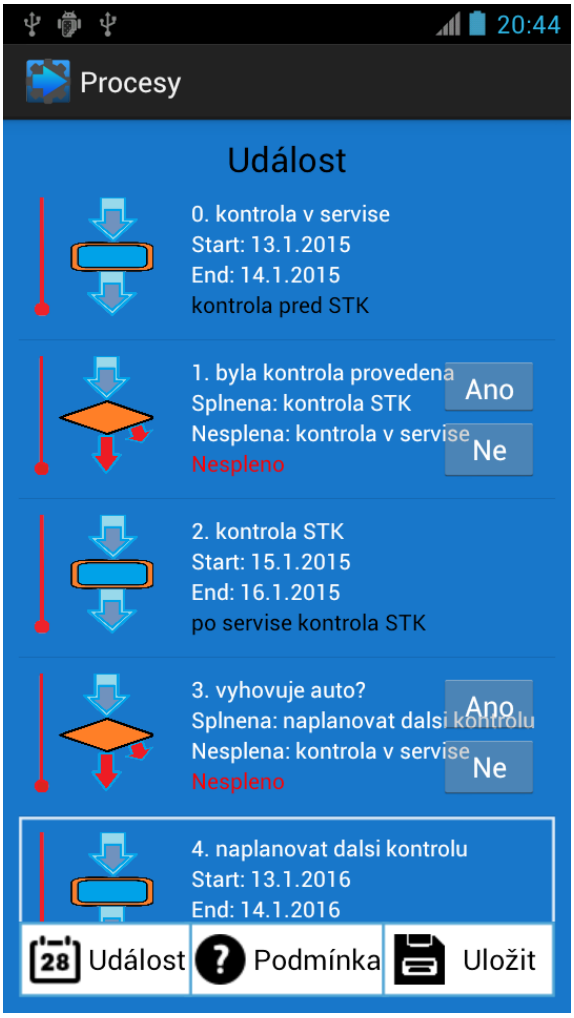
- název a popis - *EditText*
- splněna a nesplněna podmínka - *Spinner*

4.4.6 Třída ProcesSetAdapter a EventHolder

Další třída, která je důležitá pro chod aplikace je třída, které dědí z *AdapterView*. Adaptér poskytuje přístup k datovým položkám, které jsou zobrazeny například ve formě seznamu (*ListView*). Adaptér je také zodpovědný za vytváření zobrazení pro každou položku v souboru dat. V této aplikaci slouží třída *ProcesSetAdapter* pro zobrazení již vytvořeného a uloženého procesu - ukazuje tedy přehledně jednotlivé události a podmínky.

Využívá při tom statickou třídu *EventHolder*, která definuje jednu samostatnou položku v *ListView*. Třída *EventHolder* obsahuje tedy prvky typu *TextView* pro zobrazení popisu, například název události, *ImageView* pro znázornění uživatelsky více přívětivého obrázku a *Button* pro potvrzení podmínky.

Třídy typu *ProcesSetAdapter*, obsahuje aplikace celkem tři. Třída *DisplayAdapter* slouží pro zobrazení seznamu uložených procesů, třída *ProcesAdapter* zase pro zobrazení jednoho právě vytvářeného procesu - tedy jednotlivé události nebo podmínky.



Obrázek 15: ListView nadefinovaný pomocí vlastního adaptéru

4.4.7 Třída Proces a DisplayProcesActivity

Třída *DisplayProcesActivity* slouží pro prohlížení v seznamu všech procesů, z kterého si uživatel může libovolně vybrat. Třída obsahuje metodu *showPopup(View v)*, která umožňuje při dlouhém stisku vybrat položku z popup menu. Dále mimo jiné obsahuje odkaz na objekt třídy *Proces*.

Třída *Proces* si udržuje seznam procesů, pole jmen jednotlivých procesů a odkaz na objekt typu *TableModulP*. Dále pomocí funkce *Delete(String nameP)* může mazat procesy z paměti. Procedura *getListEvent(int id)* slouží pro vrácení seznamu událostí pro daný proces dle jedinečného identifikátoru id.

4.4.8 Třída MyCalendar

Tato třída, jak již její název napovídá, slouží pro synchronizaci s Google kalendářem. Obsahuje metody *Insert()* pro vkládání nové události, *Update()* pro její editaci a *Delete()* pro mazání [4].

4.4.9 Třída Variable, VariableValue, ChangeEditVariableActivity a EditVariableActivity

Tyto třídy obsahuje balíček *EditVariable* a slouží pro úpravu stávajících nebo přidání nových proměnných. Třída *Variable* obsahuje základní informace o proměnné a to jedinečné identifikační číslo id, název, starou a novou hodnotu a metody pro přístup k těmto údajům. Třída *VariableValue* se stará o práci s proměnnými (například načítání a ukládání z/do paměti, vkládání nových a jiné). Aktivitu *ChangeEditVariableActivity* využívá uživatel, když chce některou z již uložených proměnných změnit, vložit novou nebo smazat. Zobrazí tedy seznam všech proměnných. Aktivita *EditVariableActivity* pak slouží pro samotnou editaci proměnných (zobrazí uživateli starou hodnotu, která bude nahrazena nově zadanou) [5].

4.5 Sekvenční diagram

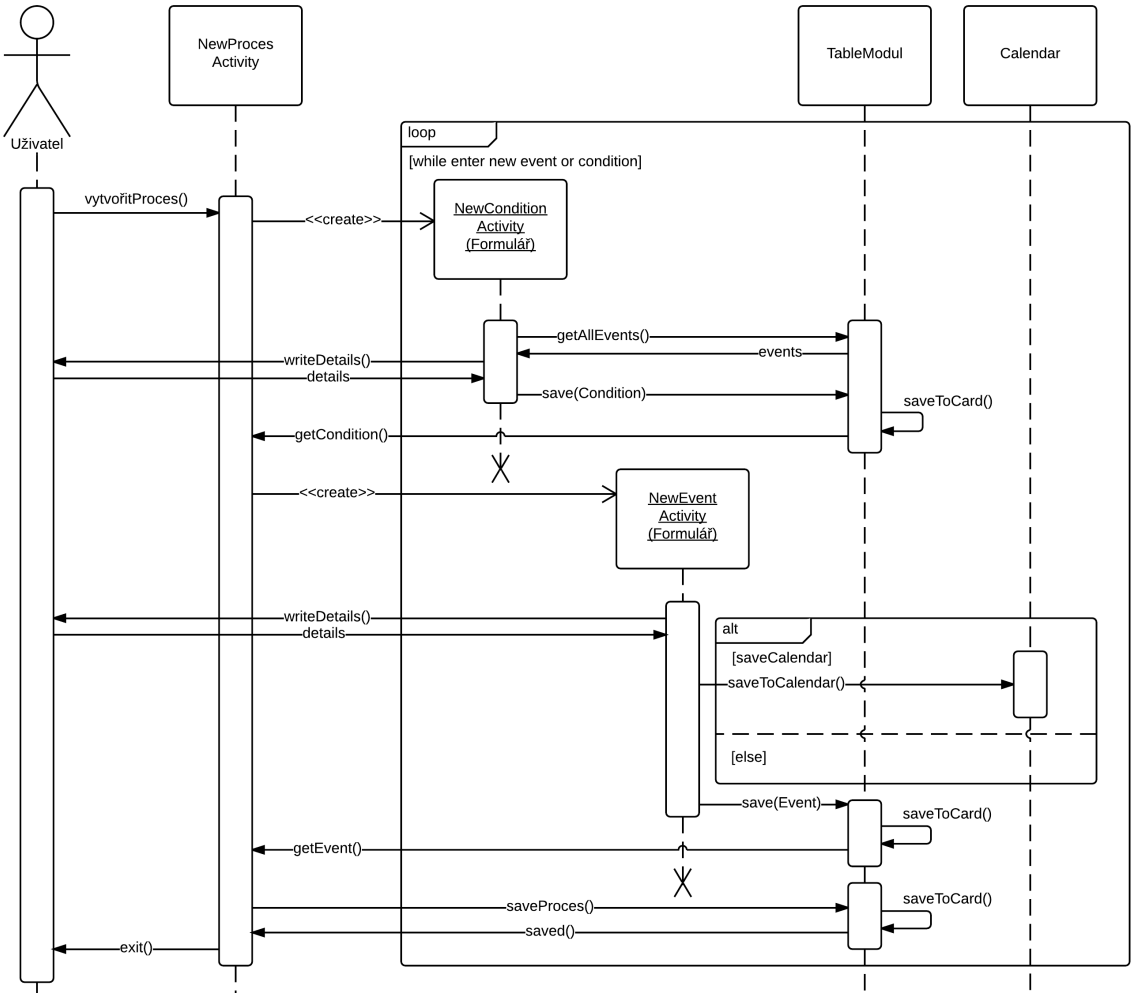
Dalším důležitým diagramem pro zobrazení interakcí mezi jednotlivými objekty a lepším pochopením fungování aplikace slouží sekvenční diagram. Případem, který stojí za důkladnější objasnění je vytváření nového procesu (pro upřesnění se jedná o případ užití UC-003 z *obrázku 13*).

Na *obrázku 16* je znázorněna komunikace mezi jednotlivými třídami a objekty při vytváření nového procesu. Při vytváření nového procesu se uživateli nejdříve zobrazí aktivita *NewProces*, která postupně zobrazuje vykonané kroky - tedy jednotlivé události a podmínky v seznamu (*ListView*). Nyní si může uživatel vybrat, zda vloží novou událost nebo podmínku. V každém případě se uživateli zobrazí nová aktivita ve formě formuláře, do kterého postupně vyplňuje údaje. Při volbě nové události má uživatel možnost zvolit si, zda se daná událost bude ukládat do kalendáře. Když uživatel vloží novou podmínku, vybírá cestu, kam bude proces pokračovat při splnění či nesplnění podmínky. Jakmile uživatel daný formulář potvrdí, uloží se událost, respektive podmínka do objektu třídy *TableModul*. Tato třída se následně postará o uložení namodelovaného procesu na SD kartu mobilního zařízení a poté vrátí uložený objekt aktivitě *NewProces*, která se postará o vykreslení. Tento proces se cyklicky opakuje, dokud uživatel přidává nové objekty. Po uložení procesu je uživatel přesunut do hlavní nabídky aplikace.

Podobný postup nastává při modifikaci procesu. Rozdíl je pouze v tom, že celý proces je nejdříve načten z SD karty. Následně jej může uživatel libovolně modifikovat a

[4] Třída MyCalendar není pro přehlednost uvedena v třídním diagramu.

[5] Třídy Variable, VariableValue, ChangeEditVariableActivity a EditVariableActivity nejsou pro přehlednost uvedeny v třídním diagramu.



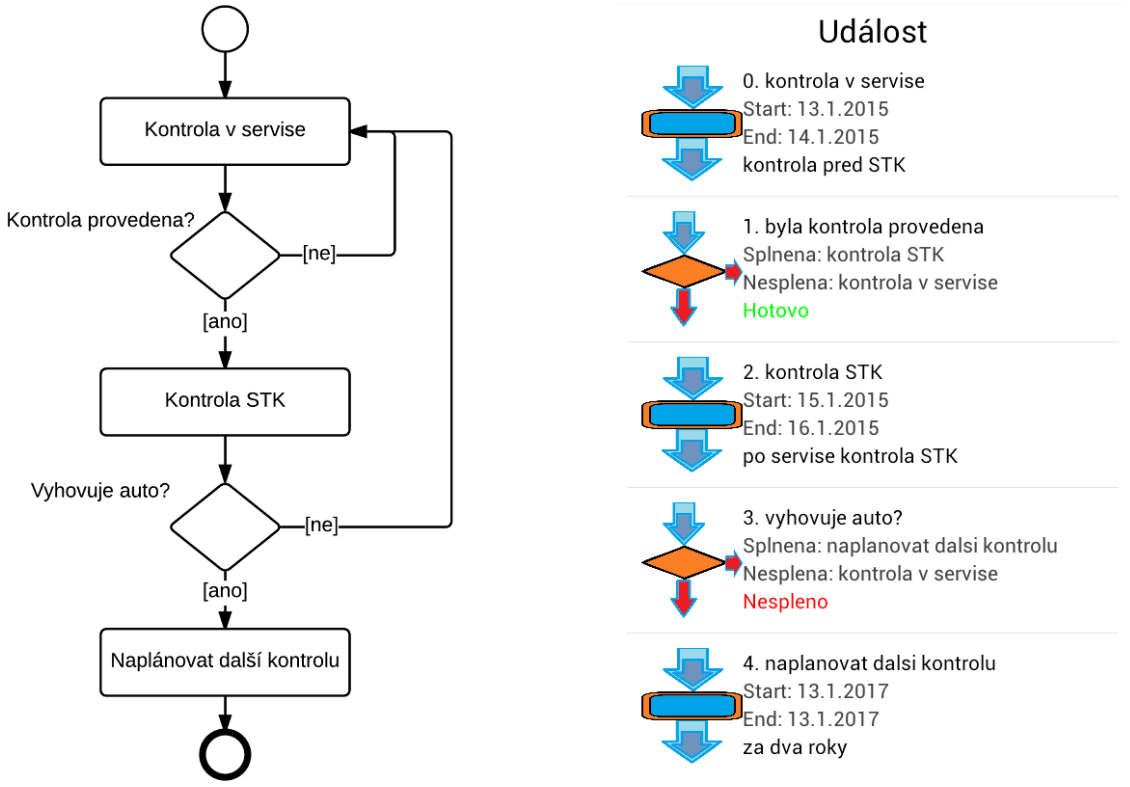
Obrázek 16: Sekvenční diagram - vytvoření nového procesu

také přidávat nové událost nebo podmínky. Volbou uložit se upravený proces nahraje na paměťovou kartu telefonu.

4.6 Vývojový diagram

Tímto diagramem jsem se v aplikaci pokusil modelovaný proces co nejlépe zachytit tak, aby se uživatel rychle a správně zorientoval. Myslím si, že u mobilní aplikace by právě modelování procesů mělo být co možná nejsnazší, protože ve většině případů se budou modelovat jednoduché procesy, které lze právě vývojovým diagramem snadno zachytit.

Dalším důvodem je interpretace na "malém"displeji mobilního zařízení, která by měla být přehledná. Na *obrázku 17* je znázorněn, jak modelovaný proces vypadá (obrázek vpravo) a jak vypadá namodelovaný proces v telefonu pomocí *ListView* (obrázek vlevo).



Obrázek 17: Vývojový diagram - vpravo schéma diagramu vytvořeného pomocí aplikace - obrázek vlevo

5 Tvorba mobilní aplikace na základě návrhu

Jelikož mám již vytvořený detailní návrh aplikace, který je popsaný v kapitole výše, může začít samotná tvorba. Ze specifikace požadavků a diagramu případu užiti na *obrázku 13* vyplývá několik důležitých informací. Je jasné, že budeme využívat Google kalendář. Další důležitou funkcí aplikace musí být určitě ukládání do paměti v takové formě, aby bylo jednoduché načítat data zpět. Aplikace by měla uživatele nějakým způsobem upozornit na změny, zde využijeme notifikace. Další důležitou vlastností je počítání výpočtů pracující s proměnnými, které si může uživatel upravovat.

5.1 Synchronizace s Google kalendářem

Aplikace sice umožňuje namodelovat nový proces, ale uživatel by měl mít také možnost zobrazit si informace přehledně, nejlépe v již existujícím a uživatelsky známém programu - Google kalendáři. Tato aplikace zobrazuje kalendář po dnech, týdnech a měsících a také agendu - tedy stručný přehled událostí v kalendáři. Navíc většina uživatelů se systémem Android má tuto aplikaci již nainstalovanou defaultně. Právě z těchto důvodů jsem zvolil synchronizaci s touto aplikací.

Práce s kalendářem od společnosti Google je velice snadná a dobře zdokumentována vývojáři [6]. Využívá se třídy *Calendar Provider*, která slouží jako úložiště pro události od daného uživatele. *Calendar Provider API* umožňuje provádět dotazy a základní operace s událostmi a upomínkami (vkládání, úpravu a mazání).

Abychom mohli využívat služeb kalendáře, to znamená číst nebo zapisovat data, musíme nejdříve povolit oprávnění pro tyto operace v souboru *AndroidManifest*, jak lze vidět z ukázky kódu 2.

```
<?xml version="1.0" encoding="utf−8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.jakub.calendar" >

    <uses−permission android:name="android.permission.READ_CALENDAR" />
    <uses−permission android:name="android.permission.WRITE_CALENDAR" />
     ...
</manifest>
```

Výpis 2: Povolení pro čtení a zápis z/do kalendáře v AndroidManifest.xml

Calendar Provider využívá k získání dat *Content Provider* (ukládá a zpřístupňuje data aplikacím ve formě tabulek, kde každý řádek je jeden záznam a každý sloupec jsou data určitého typu a významu). Skrze *Calendar Provider API* mohou aplikace získat přístup pro čtení nebo zápis do databázových tabulek, které ukládají data od uživatele v kalendáři.

Každý *Content Provider* zpřístupní veřejnou URI (zabalenou jako Uri objekt), který jednoznačně identifikuje sadu dat. *Calendar Provider* definuje konstanty pro URI, které od-

[6] Dokumentace je dostupná na internetových stránkách http://developer.android.com/guide/topics/providers/calendar-provider.html (2. 4. 2015)

povídají každé ze svých tříd (tabulek). Tyto URI mají formát `<class>.CONTENT_URI` (například `Events.CONTENT_URI`).

Pro dotaz, který hledá kalendáře, musíme zvolit *ACCOUNT_NAME*, *ACCOUNT_TYPE* a *OWNER_ACCOUNT*. V ukázce kódu 3 vracejí statické metody *getNameEmail()* a *getTypeEmail()* email a typ emailu uložených v nastavení pomocí třídy *SaveEmail*. Dotaz vrátí objekt typu *Cursor*, který můžeme použít k procházení výsledků vrácených z dotazu databáze.

```
    ...
Uri uri = CalendarContract.Calendars.CONTENT_URI;
String selection = "((" + CalendarContract.Calendars.ACCOUNT_NAME + "␣=␣?)␣AND␣("
                + CalendarContract.Calendars.ACCOUNT_TYPE + "␣=␣?)␣AND␣("
                + CalendarContract.Calendars.OWNER_ACCOUNT + "␣=␣?))";

String [] selectionArgs = new String[] {SaveEmail.getNameEmail(), SaveEmail.getTypeEmail(),
    SaveEmail.getNameEmail()};
cur = cr.query(uri, EVENT_PROJECTION, selection, selectionArgs, null);
    ...
```

Výpis 3: Ukázka synchronizace s kalendářem přes email a typ uživatele.

Poté se už jednoduše vloží potřebné údaje pomocí *ContentValues* do kalendáře přes vytvořené URI. Nakonec vracíme hodnotu *eventID*, tedy identifikační číslo právě vytvořené události v kalendáři, přes které můžeme tuto událost upravovat nebo ji zcela smazat. Ukázka kódu 4 popisuje funkci *Insert* v třídě *MyCalendar*, obdobně pracuje i funkce *Update* s parametrem ID události.

```
    ...
startMillis = startDate.getTimeInMillis();
endMillis = endDate.getTimeInMillis();

ContentResolver contentResolver = context.getContentResolver();
ContentValues values = new ContentValues();
values.put(CalendarContract.Events.DTSTART, startMillis);
values.put(CalendarContract.Events.DTEND, endMillis);
values.put(CalendarContract.Events.TITLE, name);
values.put(CalendarContract.Events.DESCRIPTION, description);
Uri uri = contentResolver.insert(CalendarContract.Events.CONTENT_URI, values);

long eventID = Long.parseLong(uri.getLastPathSegment());

return (int)eventID;
    ...
```

Výpis 4: Funkce Insert (třída MyCalendar).

5.2 Ukládání na SD kartu ve formátu JSON

Důležitou funkcí pro opakované používání aplikace je nepochybně ukládání, ať už nových nebo upravených procesů a proměnných. Jelikož by aplikace mohla být dále rozšiřitelná a uložená data by mohla být přístupná i pro jiné aplikace, rozhodl jsem se ukládat

data na SD kartu v mobilním zařízení.

Při využívání možností zapisovat/číst data z externího úložiště je potřeba požádat o povolení `WRITE_EXTERNAL_STORAGE` respektive `READ_EXTERNAL_STORAGE` v *Android Manifestu* podobně jako při práci s kalendářem 2. Vždy při ukládání nebo čtení dat z SD karty zjišťuji stav připojení SD karty v zařízení, jak lze vidět v ukázce kódu 5. Metoda *WriteToFile* přijímá jediný parametr typu *String*, který obsahuje řetězec naformátovaný pomocí JSON.

```
public void WriteToFile(String listJSONString){
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)){
        File sdCard = Environment.getExternalStorageDirectory();
        ...
          File file = new File(dirIn, "variables.txt");
              try{
                   FileWriter fw = new FileWriter( file );
                   BufferedWriter bw = new BufferedWriter(fw);
                   bw.write(listJSONString);
                   bw.flush();
                   bw.close();
              }catch (Exception e){
                   e.printStackTrace();
              }
    }
}
```

Výpis 5: Metoda WritToFile.

5.2.1 JSON

Pro vkládání údajů na SD kartu zařízení jsem si zvolil uložení ve formátu JSON. JSON (JavaScript Object Notation) je způsob objektového zápisu dat určených pro přenos. JSON je textový formát a proto je lidsky dobře čitelný, lze jednoduše parsrovat a generovat a je nezávislý na platformě. Tyto vlastnosti dělají z JSON ideální jazyk pro výměnu dat. Vstupem je libovolná datová struktura a výstupem je vždy textový řetězec. JSON je postaven na dvou strukturách:

- Kolekce párů název/hodnota. V různých jazycích je toto realizováno jako objekt, záznam, struktura, slovník a jiné.
- Seřazený seznam hodnot. Ve většině jazyků to je realizováno jako pole, vektor, seznam nebo sekvence.

Jedná se o univerzální datové struktury. V podstatě všechny moderní programovací jazyky podporují práci s JSON. Struktura formátu JSON: objekt tvoří neuspořádané dvojice název/hodnota. Objekt začíná `{` a končí `}`. Každý název je následovaný : a páry název/hodnota jsou odděleny , (čárkou). Pole je setříděná kolekce hodnot. Pole začíná [a končí]. Hodnoty jsou také odděleny , (čárkou). Hodnota může být řetězec v dvojitých

uvozovkách, číslo, *true*, *false*, *null*, objekt nebo pole. Struktury mohou být zanořené a to umožňuje velké možnosti.

```
    ...
JSONArray list = new JSONArray();
JSONObject objectToSave = new JSONObject();
    try{
        for(int i = 0; i < ListVariable.size(); i++){
            JSONObject obj = new JSONObject();
            obj.put("id", ListVariable.get(i).getId());
            obj.put("name", ListVariable.get(i).getName());
            obj.put("oldValue", ListVariable.get(i).getOldValue());
            obj.put("newValue", ListVariable.get(i).getNewValue());
             list.put(obj);
           }
           objectToSave.put("variables", list);
           objectToSave.put("nameJSON", "Variable");
    }catch (JSONException ex){
           ex.printStackTrace();
     }
String listJSONString = objectToSave.toString();
    ...
```

Výpis 6: Ukládání dat do objektu JSON.

Ve výpisu kódu 6 nejprve vytvářím list typu *JSONArray*, do kterého postupně vkládám objekty typu *JSONObject*. Tento objekt vzniká uvnitř cyklu, který postupně prochází všechny proměnné, a plní se jednotlivými daty (id, jméno, stará a nová hodnota) právě prohlíženou proměnnou. Výsledek lze vidět v ukázce 7.

```
{"nameJSON":"Variable",
    "variables":[
        {"id":0,"newValue":32.2,"oldValue":35.2,"name":"benzin"},
        {"id":1,"newValue":29.6,"oldValue":30.2,"name":"nafta"}
    ]
}
```

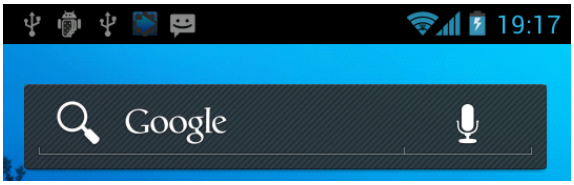
Výpis 7: Uložené informace ve formátu JSON.

5.3 Práce s notifikacemi

Nespornou výhodou při práci s aplikací budou pro uživatele upozornění v podobně notifikací. Tato připomenutí budou zobrazovat začátek právě probíhající události a umožňovat uživateli na ně reagovat i při vypnuté aplikaci.

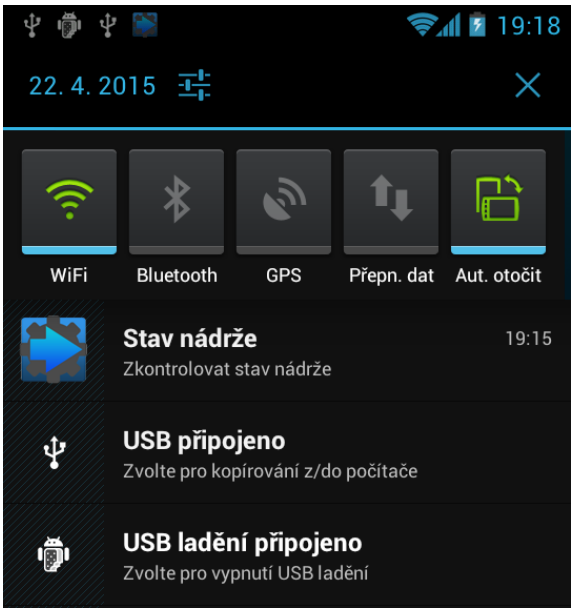
5.3.1 Notifikace

Notifikace je zpráva, která se uživateli může zobrazit, aniž by právě používal aplikaci, která ji vyvolala. Když se notifikace vyvolá, zobrazí se v oznamovací ("notifikační") oblasti *(obrázek 18)*.



Obrázek 18: Oblast notifikací

Abychom mohli vidět detaily oznámení, musí uživatel srolovat tuto oznamovací lištu *(obrázek 19)*. Obvykle se při kliknutí na tuto notifikaci uživatel přesune do aplikace.



Obrázek 19: Jednotlivé notifikace v notifikační liště

5.3.2 Broadcast Receiver

Abychom mohli používat notifikace, musíme využít služeb *Broadcast Receivera*. *Broadcast Receiver* je prvek systému Android, který nám umožní zaregistrovat se k systému. Všechny zaregistrované přijímače *Receivers* pro nějakou událost jsou oznámeny androidu za běhu, jakmile je tato událost vyvolána. Například registrování systémové události `ACTION_BOOT_COMPLETED`, která je spuštěna jednou při dokončení "bootování"systému. Vytvářená třída pro implementaci přijímače musí rozšiřovat třídu *BroadcastReceiver*. V případě, že nastane událost, kterou má přijímač zaregistrovanou, dojde ke spuštění metody *onReceive*, která je volána napříč systémem.

5.3.3 Použití

Při první implementaci jsem využil služeb poskytující třídou *Service*. Tato služba běží neustále na pozadí a proto může vyvolat notifikace a zároveň provádět další činnost. Setkal jsem se ovšem s náhlými pády aplikace, které je těžké ladit a tak nalézt problém. Nakonec jsem od využívání služeb upustil, protože k používání notifikací nejsou nezbytně nutné. Pro možnost spustit danou notifikaci v uživateli zvoleném čase jsem využil *AlarmManager*. Tato třída poskytuje přístup k systémovým "alarm services". Při spuštění alarmu je vyvolán *Intent*, který je registrovaný a dojde k automatickému spuštění cílové aplikace. Pro využívání služeb *AlarmManager* musíme v *Android Manifestu* požádat o povolení `WAKE_LOCK`. V ukázce kódu 8 je zobrazena metoda *onReceive* ve třídě *MyReceiver*, která dědí z třídy *BroadcastReceiver*. Uvnitř metody se pomocí *Intentu* získají parametry (jméno, popis a identifikační číslo) a dojde k vytvoření notifikace a následnému zobrazení.

```
    ...
String name = intent.getStringExtra("name");
String description = intent.getStringExtra("description");
int id = intent.getIntExtra("id", 1);

Intent i = new Intent(context, MainActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, i, PendingIntent.
    FLAG_UPDATE_CURRENT);
Notification notification = NotifyUtils.createNotification(context, name, description,
    pendingIntent);
NotifyUtils.showNotification(context, notification, id);
    ...
```

Výpis 8: Metoda onReceive.

5.4 Modelování nového procesu

Princip modelování nového procesu je detailně popsán v kapitole 4.5 Sekvenční diagram a na sekvenčním diagramu 16. Funkcí, která zvyšuje uživatelský komfort je bezpochyby *repeatingWeek* v ukázce 9.

```
    ...
Calendar firstDay = Calendar.getInstance();
firstDay.set(startYear, startMonth, startDay, startHour, startMinute);
Calendar secondDay = Calendar.getInstance();
secondDay.set(endYearRepeating, endMonthRepeating, endDayRepeating, endHour, endMinute);
int countDays = (int)((secondDay.getTimeInMillis() - firstDay.getTimeInMillis()) /
    (1000*60*60*24));
if (countDays < 6){
   Toast.makeText(context, "Spatna hodnota!", Toast.LENGTH_SHORT).show();
    return;
}
    ...
for(int i = 0; i <= countDays; i = i + 7){
   startDate.add(Calendar.DATE, tmp);
```

```
endDate.add(Calendar.DATE, tmp);
...
if (saveCalendar) {
  MyCalendar myCalendar = new MyCalendar();
    idCalendar = myCalendar.Insert(cr, cur, context, name, description, startDate, endDate,
        allDay);
}
 ...
```

Výpis 9: Metoda repeatingWeek.

Tato funkce slouží pro opakující se události každý týden (implementovány jsou rovněž funkce *repeatingMonth* pro opakování co měsíc a *repeatingDay* pro opakování každý den). Funkce odečte dvě data (konečné a počáteční zadané datum) a vydělí počtem milisekund za jeden den. Pokud je výsledný počet dní menší než 6, zobrazí se uživateli zpráva o špatně zadané hodnotě. V opačném případě se v cyklu projdou jednotlivé týdny a vytvoří se k nim patřičné události. Tato funkce je vhodná například pro vyřizování každodenních úkonů nebo při měsíčních kontrolách stavu vozidla.

5.5 CalculateActivity

Nezanedbatelnou funkcí aplikace jsou různé výpočty, které usnadní uživateli každodenní činnosti. Konkrétně u profese taxikáře jsem implementoval výpočty pro spotřebu paliva, náklady vozidla na kilometr, výpočet zisku a výpočet platu. Tyto výpočty pracují s uloženými proměnnými, které lze měnit a tím ovlivňovat výsledky výpočtů. Kdybychom chtěli aplikaci použít pro jinou profesi, v podstatě by stačilo vyměnit pouze tuto část, která je specifická pro danou profesi.

```
case 3:
   try{
        double startKM = Double.parseDouble(editTxtValue1.getText().toString());
         ...
        double totalKM = endKM − startKM;

        if (totalKM > 0 && countLitres > 0){
            double average = (countLitres/totalKM) ∗ 100;
            double result = (countLitres/totalKM) ∗ costValue;
            double resultMoney = money − (result ∗ totalKM);
            txtViewResult.setText(String.format("Prumerna␣spotreba:␣%.2f␣litru/100km.␣\n" +
            "Naklady:␣%.2f␣kc/km.␣\n" +
            "Cisty␣zisk:␣%.2f␣kc.", average, result, resultMoney));
         }
   }catch (Exception e){
         e.printStackTrace();
         Toast.makeText(getApplicationContext(), "Spatne␣zadane␣hodnoty!", Toast.
              LENGTH_SHORT).show();
   }
   break;
```

Výpis 10: Část výpočetní metody - výpočet průměrné spotřeby.

Jelikož je těchto výpočtů více a liší se popisem vstupních prvků a průběhu výpočtů, rozhodl jsem se tyto výpočty implementovat do jediné aktivity. V této aktivitě vždy zobrazuji nebo skrývám jednotlivé položky specifické pro daný výpočet. Poté již následuje samotný výpočet - kontrola zadaných údajů, výpočet a zaokrouhlení pro uživatele snadnější čtení, jak lze vidět v ukázce 10.

6 Závěr

Cílem této bakalářské práce bylo vytvoření mobilní aplikace pro podporu procesů specifické profese, a to taxikáře. Aby bylo možné takovouto aplikaci vytvořit tak, aby splňovala požadavky pro tuto charakteristickou profesi, bylo potřeba provést detailní analýzu a návrh aplikace. Během analýzy jsem specifikoval požadavky, které by měla aplikace splňovat a tím ulehčit plánování a řízení činnosti taxikáře. Mezi tyto požadavky patří mimo jiné využívání již namodelovaných procesů, vytváření vlastních procesů, využívání synchronizace s kalendářem, využívání notifikací, použití výpočtů a editace proměnných. Po důkladné analýze následoval detailní návrh aplikace, který obsahoval diagram tříd nutných k základní funkci aplikace. Poté jsem vytvořil sekvenční diagram, který ukazuje, jak probíhá modelování nového procesu z pohledu objektů.

Po důkladné analýze a návrhu jsem začal přemýšlet nad platformou, pod kterou bude aplikace fungovat. Zhodnotil jsem tedy situaci na trhu, při které jsem zjistil, že většina chytrých mobilních telefonů využívá systém Android a drtivá většina verzi 4+. Navíc je vyvíjení pro tuto platformu zdarma, kromě registračního poplatku do Google Play. Rozhodl jsem se tedy vyvíjet aplikaci pro systém Android. Dalším otazníkem bylo rozhodnutí při výběru vývojového prostředí. Zvolil jsem IDE Android Studio, které je dnes již bráno jako standard, a které umožňuje snadnou instalaci a intuitivní režim tvorby aplikací.

Jelikož jsem měl návrh aplikace hotový a vybraný systém, pro který budu vyvíjet, následovala samotná implementace. Postupoval jsem podle návrhu a začal podle třídního diagramu. Při vývoji pro systém Android jsem postupně narazil na několik odlišností, mezi které patří třída Activity a soubor AndroidManifest. Na další jsem postupně přicházel při vývoji, například využívání služeb poskytujících přístup ke kalendáři přes Calendar Provider, ukládání namodelovaného procesu na externí zařízení ve formátu JSON. Důležitou funkčností byla také možnost upozornit uživatele na právě probíhající událost formou notifikace, kde bylo zapotřebí využít služeb Broadcast Receiver a Alarm Manager, který zobrazí notifikaci ve správný čas. V neposlední řadě jsem vytvořil základní výpočty, důležité pro tuto profesi pracující s proměnnými, které lze libovolně upravovat či vytvářet.

Dalším bodem této práce bylo také seznámit čtenáře s úvodem do procesního řízení. Tuto kapitolu jsem vytvářel s důrazem na jazyk UML, který jsem používal při návrhu aplikace. Využil jsem různé typy diagramů, například třídní, vývojový, sekvenční, diagram případu užití a další, které jsem v této části detailně popsal a vysvětlil jejich význam.

Součástí bakalářské práce bylo také vytvořit uživatelskou příručku, která detailně a přehledně ukazuje základní funkce aplikace. Příručka by se později mohla stát součástí aplikace v podobě nápovědy. Aplikaci jsem vytvářel s ohledem na to, aby jí bylo snadné použít i pro jiné profese než taxikáře. To se mi vcelku podařilo. Jediné co nelze uplatnit pro jiné povolání jsou výpočty, které jsou charakteristické právě pro tuto profesi. Také jsem se snažil vytvořit aplikaci s možností jejího dalšího rozvoje, kde především využití formátu JSON pro ukládání nových procesů by rozvoj mohlo velmi usnadnit. Jelikož

jsem splnil určené požadavky při analýze a implementoval jsem funkce důležité pro tuto profesi, byl cíl této bakalářské práce splněn.

7 Reference

[1] F. Sowa, John, *Knowledge representation: logical, philosophical, and computational foundations, Pacific Grove, California: Brooks Cole Publishing Co., 1999, 608s. ISBN 0 534-94965-7*

[2] Wei-Meng, Lee, *Beginning Android 4 Application Development, Wrox Press, 2012, 560s, ISBN 978-1-118-19954-1*

[3] ARIS, *Procesní řízení* [online], [cit. 2015-2-22], dostupné z: http://www.arisys.cz/inpage/isrpro3/

[4] *Definice procesu* [online], 17. 3. 2008, [cit. 2015-2-22], dostupné z: http://zcu.arcao.com/kpv/pis/pinte/Definice_procesu_dle_EN_ISO_9000-2000.pdf

[5] Management mania, *Podnikový proces* [online], 10. 1. 2015, [cit. 2015-2-22], dostupné z: https://managementmania.com/cs/business-process-podnikovy-proces

[6] Metody byznys modelování, *Byznys modelování pomocí UML* [online], Ivo Vondrák, 5. 9. 2004, [cit. 2015-2-23], dostupné z: http://vondrak.cs.vsb.cz/download/Metody_byznys_modelovani.pdf

[7] Příklady použití diagramů UML 2.0, *Stavový diagram, diagram aktivit, diagram případů užití, sekvenční diagram* [online], Petra Rejnková, 2009, [cit. 2015-2-23], dostupné z: http://uml.czweb.org/index.html

[8] IBM, *Business Process Management* [online], [cit. 2015-2-23], dostupné z: http://www-03.ibm.com/software/products/cs/category/BPM-SOFTWARE

[9] IXTENT, *Business Process Management* [online], 6. 1. 2015, [cit. 2015-2-23], dostupné z: http://www.ixtent.com/cs/produkty-a-reseni/process-management-workflow/business-process-management.html

[10] ITnetwork.cz, *1. díl - Úvod do UML, 5. díl - UML - Class diagram* [online], David Čápka, [cit. 2015-3-3], dostupné z: http://www.itnetwork.cz/uml-uvod-historie-vyznam-a-diagramy

[11] mpavus.wz.cz, *Class diagram - diagram tříd, Use case diagram - diagram případů užití, Sequence diagram - sekvenční diagram* [online], 18. 7. 2006, [cit. 2015-3-3], dostupné z: http://mpavus.wz.cz/uml/uml-uvod-1.php

[12] Principy objektově orientovaného programování, *Agregace a kompozice* [online], Miroslav Beneš, [cit. 2015-3-3], dostupné z: http://www.cs.vsb.cz/benes/vyuka/upr/texty/objekty/ch01s02s03.html

[13] Zdroják.cz, *Android Studio – nové vývojové prostředí* [online], Vojtěch Semecký, 4. 11. 2013, [cit. 2015-3-14], dostupné z: http://www.zdrojak.cz/clanky/android-studio-nove-vyvojove-prostredi/

[14] Zdroják.cz, *Vyvíjíme pro Android: První krůčky* [online], Matěj Konečný, 22. 6. 2012, [cit. 2015-3-17], dostupné z: http://www.zdrojak.cz/clanky/vyvijime-pro-android-prvni-krucky/

[15] developer.android.com, *Android Studio, Android Studio Overview, Using the Emulator, Activity, Debugging with Android Studio, App Manifest, Notifications* [online], [cit. 2015-3-14], dostupné z: http://developer.android.com/sdk/index.html

[16] petrhouf.blogspot.cz *Vývoj pro Android II.* [online], Petr Houf, 7. 10. 2013, [cit. 2015-3-17], dostupné z: http://petrhouf.blogspot.cz/2013/10/pripravazarizeni.html

[17] json.org, *Introducing JSON* [online], [cit. 2015-4-2], dostupné z: http://json.org/

[18] Vogella, *Android BroadcastReceiver - Tutorial* [online], Lars Vogel, 14. 8. 2013, [cit. 2015-4-17], dostupné z: http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html

A CD s aplikací

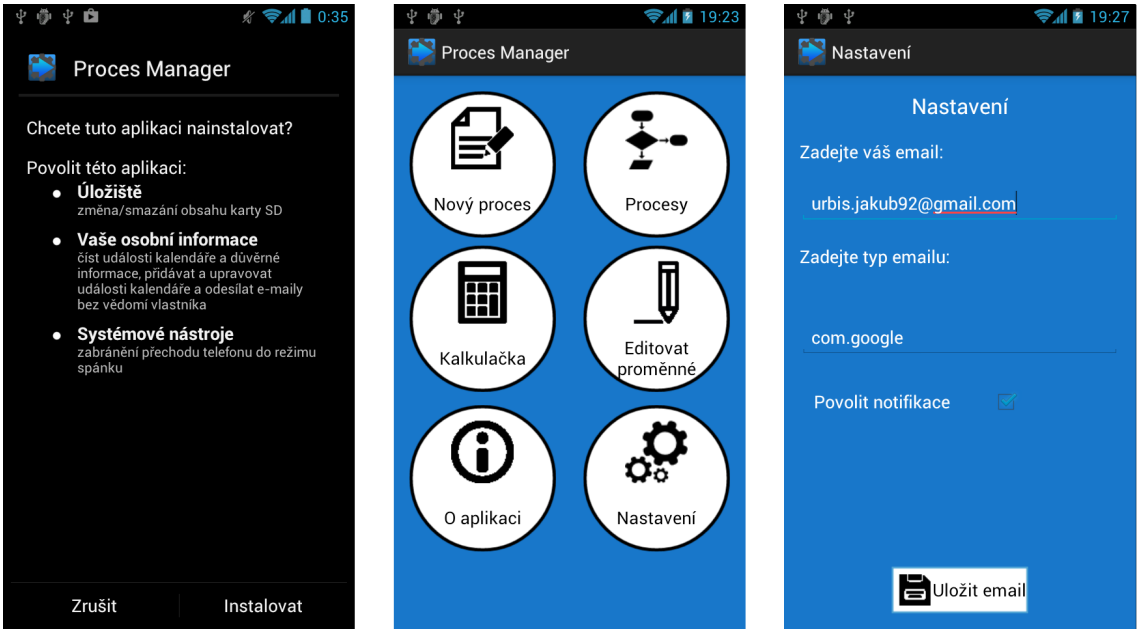
Přiložené CD obsahuje:

- vytvořenou aplikaci (instalační soubor s příponou *.apk*)
- zdrojové kódy aplikace
- Uživatelskou příručku, která by měla vysvětlit základní funkce vytvořené aplikace. Příručka má za úkol usnadnit uživateli prvotní orientaci a zacházení s touto aplikací.

B Uživatelská příručka

B.1 Hlavní menu, nastavení

Poté, co si uživatel stáhne aplikaci *Proces Manager* do svého mobilního telefonu (postup instalace a stažení je stejný jako u jiných aplikací (obrázek 20 vlevo) [7]), zobrazí se hlavní menu programu (obrázek 20 uprostřed) [8]. Abychom mohli využívat synchronizaci s aplikací Google kalendář, musíme nejdříve vyplnit *email* a *typ emailu*, který se váže k našemu účtu. Tyto informace vyplníme v sekci *nastavení* (obrázek 20 vpravo). Jestliže nechceme, aby nás aplikace upozorňovala na právě probíhající události, stačí odkliknout možnost *povolit notifikace* a kliknout na tlačítko pro uložení.



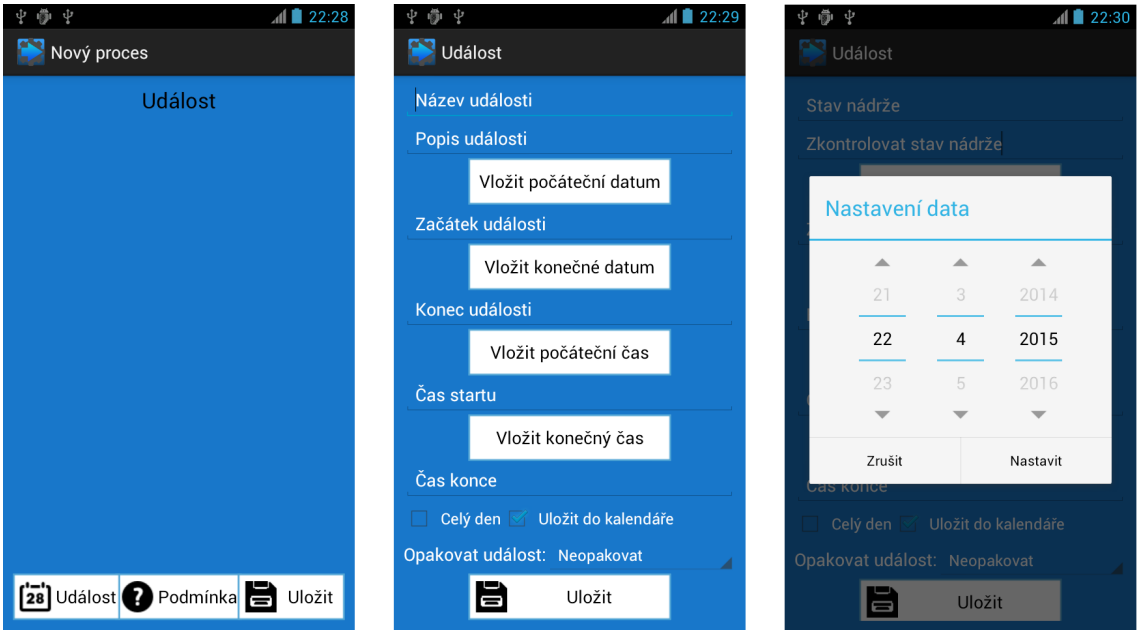
Obrázek 20: Instalace, hlavní nabídka a nastavení

B.2 Nový proces

Pokud si chceme namodelovat vlastní proces, zvolíme možnost *Nový proces* z hlavního menu. Zobrazí se nám prozatím prázdná obrazovka s třemi tlačítky a to *Událost*, *Podmínka* a *Uložit* (obrázek 21 vlevo). Pro vložení nové událost stačí kliknout na tlačítko *Událost*. Aplikace nás přesune do formuláře (obrázek 21 uprostřed), kde jednotlivé popisky napovídají, které data máme kam vložit. Vyplníme *název* a *popis události*. Po kliknutí na *vložit počáteční/konečné datum* nebo *vložit počáteční/konečný čas* se nám zobrazí dialogové okno pro vložení příslušné hodnoty (obrázek 21 vpravo). Jestliže chceme, aby

[7] Aplikace je zatím nedostupná přes internetový obchod Google Play.
[8] V aplikaci byly použity ikony zdarma ke stažení (http://www.flaticon.com/)

nás aplikace upozorňovala celý den na probíhající událost, zatrhneme možnost *Celý den*. Pro uložení události do kalendáře zvolíme *Uložit do kalendáře*. Pokud potřebujeme danou událost opakovat (například co týden) vyplníme *Opakování události*. Máme na výběr opakovat denně, týdně nebo měsíčně. Nakonec námi vytvořenou událost uložíme a aplikace nás přesune zpátky do okna *Nového procesu*, tentokrát již s vytvořenou událostí (obrázek 22 vlevo).



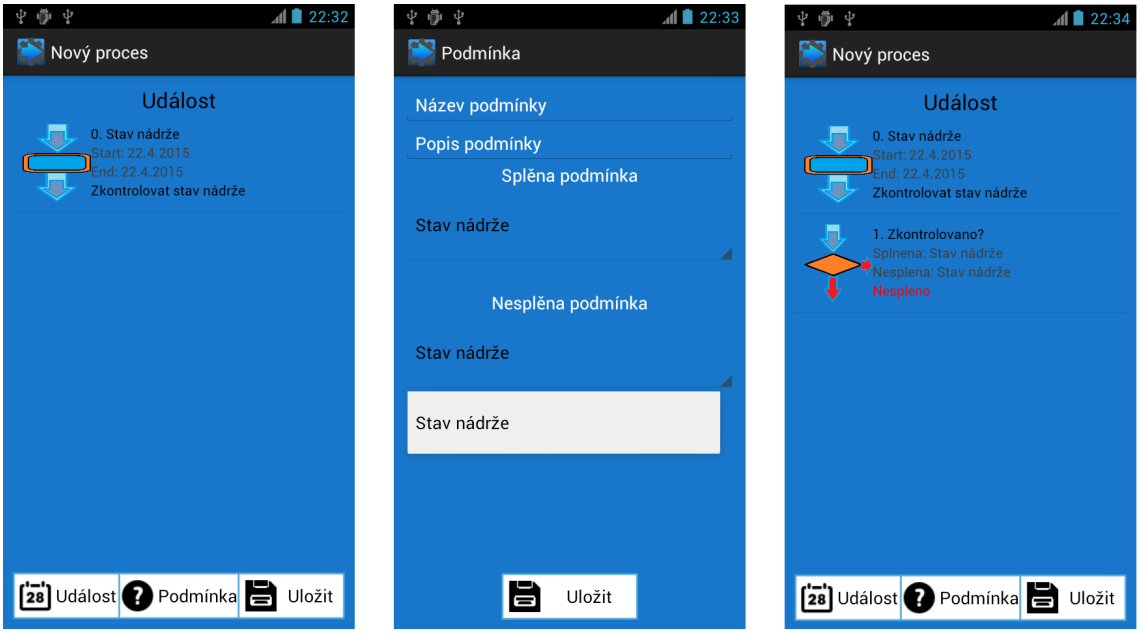
Obrázek 21: Nový proces, vytvoření události

Pro vytvoření podmínky klikneme na tlačítko *Podmínka* z obrazovky nového procesu. Přesuneme se do formuláře pro novou podmínku (obrázek 22 uprostřed) a vyplníme *Název* a *Popis podmínky*. Dále u podmínky zvolíme, kam se má postupovat v případě splnění respektive nesplnění podmínky. Aplikace nám sama nabídne již vytvořené události nebo podmínky, které máme na výběr. Po uložení podmínky se dostaneme opět do okna pro *Nový proces* (obrázek 22 vpravo). Tento proces opakujeme tak dlouho, dokud nejsme spokojeni s naším procesem. Poté stiskneme tlačítko *Uložit* (pro uložení celého procesu) a vyplníme název procesu.

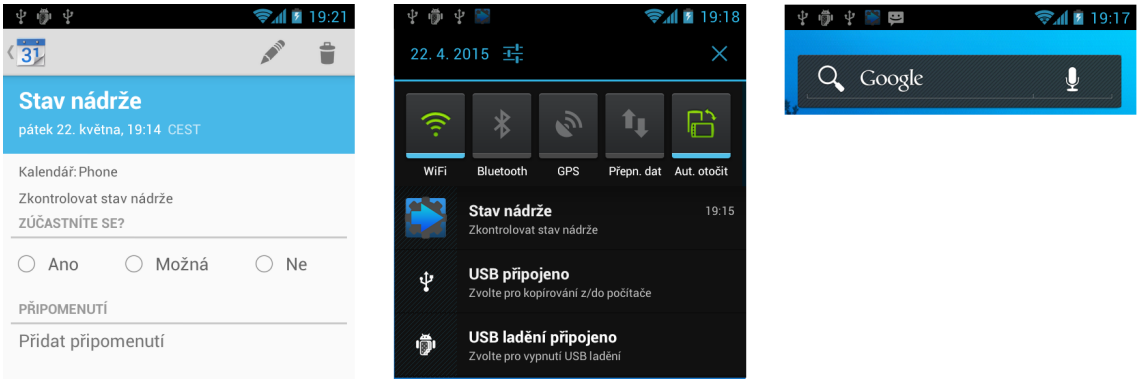
Na obrázku 23 vlevo lze vidět, jak se událost vytvořená v předchozím kroku zobrazila v kalendáři a jak na tuto událost upozornila aplikace pomocí notifikace (obrázek 23 uprostřed a vpravo).

B.3 Procesy

Pokud se chceme podívat, upravit nebo smazat již vytvořený proces, vybereme možnost *Procesy* z hlavní nabídky. Dostaneme se tak do náhledu všech existujících procesů (obrázek 24 vpravo). Při delším stisknutí prstu na vybraném procesu se nám zobrazí podna-

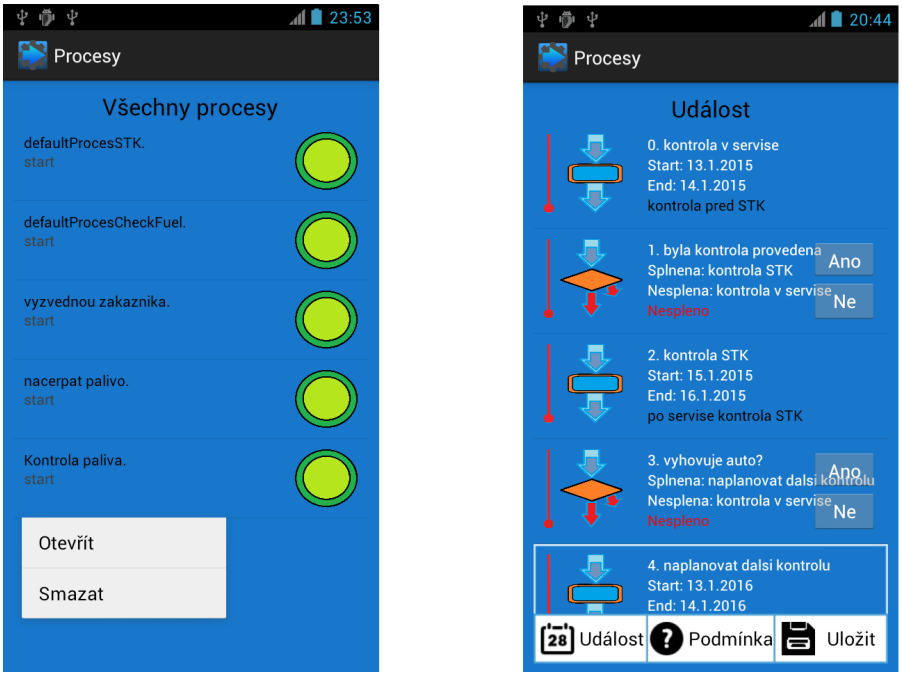


Obrázek 22: Nový proces, vytvoření podmínky



Obrázek 23: Vytvořená událost v kalendáři, notifikace upozorňující na právě probíhanou událost

bídka pro otevření nebo smazaní procesu. Při otevření se dostaneme do módu pro editaci procesu (obrázek 24 vlevo). Kliknutím na nějako položku se dostaneme opět do formuláře pro událost respektive podmínku (obrázek 21 uprostřed a 22 uprostřed). Po úpravě hodnot zvolíme uložit (podmínku, událost) a také proces. Bíle ohraničená položka značí událost, která se zatím nekonala.



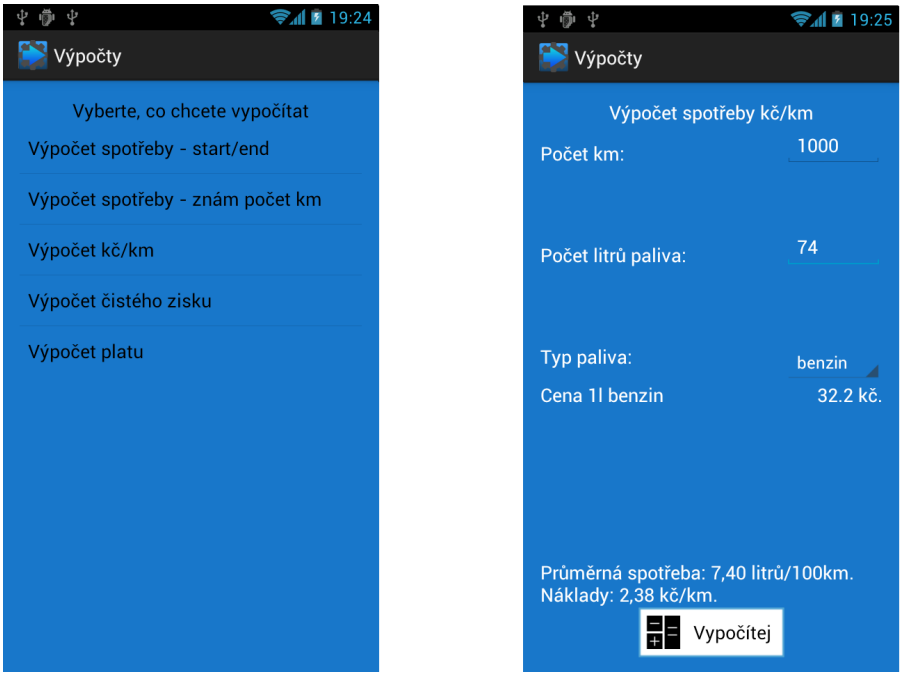
Obrázek 24: Přehled všech procesů a režim editace vybraného procesu

B.4 Kalkulačka

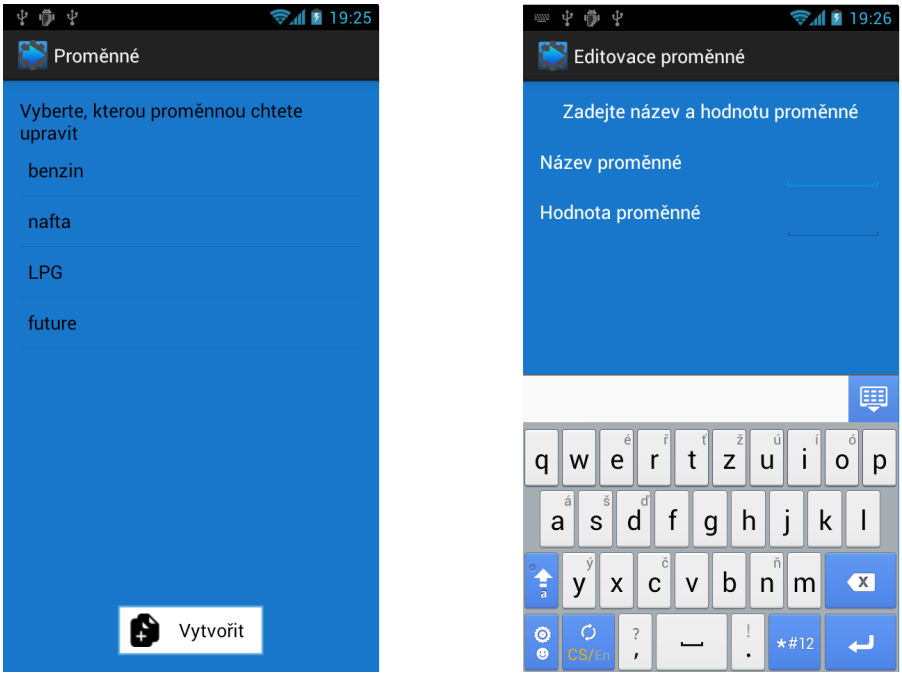
Mezi důležité a užitečné funkce patří také *Kalkulačka*. Kalkulačka umožňuje výběr z předem nadefinovaných výpočtů, které pracují s proměnnými. Pro tuto funkci zvolíme možnost *Kalkulačka* z hlavní nabídky. Poté nám aplikace nabídne výběr z několika výpočtů (obrázek 25 vlevo), z kterých si jeden zvolíme. Tímto krokem se dostaneme do formuláře pro samotný výpočet (například obrázek 25 vpravo pro výpočet průměrné spotřeby a nákladů). Po vyplnění a vybrání všech potřebných údajů nám aplikace přehledně zobrazí požadované výsledky. V ukázkovém příkladu na obrázku 25 vpravo vybíráme *Typ paliva* jako předem nadefinovanou proměnnou, kterou lze upravit.

B.5 Editace proměnných

Další důležitou vlastností aplikace je možnost vytvoření vlastních proměnných nebo editace již existujících, které se následně používají pro výpočty. K této funkci se dostaneme přes výběr *Editovat proměnné* z hlavního menu. Následně se nám zobrazí obrazovka pro výběr již existující proměnné (obrázek 26 vlevo). Pokud chceme vytvořit vlastní proměnnou, klikneme na *Vytvořit* a tím se přesuneme k tvorbě vlastní proměnné (obrázek 26 vpravo). Vypíšeme *Název* a *Hodnotu proměnné* a potvrdíme tlačítkem *Uložit*. Pro editaci proměnné vypadá obrazovka podobně jen s tím rozdílem, že zapisujeme jen novou hodnotu proměnné.



Obrázek 25: Obrazovka pro výběr výpočtu a formulář pro zvolený výpočet



Obrázek 26: Obrazovka pro vytváření, úpravu a mazání proměnné, formulář pro vložení nové proměnné